



# C7. Réseaux de neurones récurrents

Advanced Machine Learning (MLA)  
M2 Engineering of Intelligent Systems  
& Advanced Systems and Robotics

2020-2021

## 1. Introduction

- Modélisation de données séquentielles
- Exemples d'architectures et d'applications

## 2. Réseaux de neurones récurrents (RNN)

- Principe et représentation graphique
- Rétro-propagation dans le temps (BPTT)
- Disparition/explosion du gradient

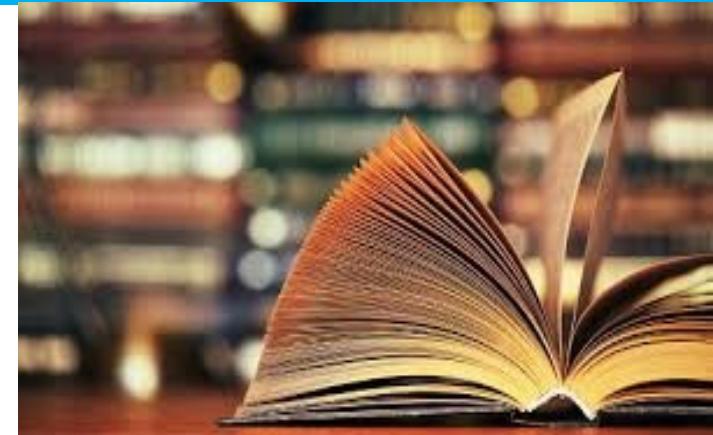
## 3. Architectures avancées

- LSTM & GRU
- Modèle séquence-à-séquence (S2S)
- Mécanisme d'attention
- Self-attention et réseaux transformers

# Conseil de lectures

## Livres et articles

Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning*. (MIT Press, 2016).



Bahdanau, D., Cho, K.-H., Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate, International Conference on Learning Representations (ICLR).

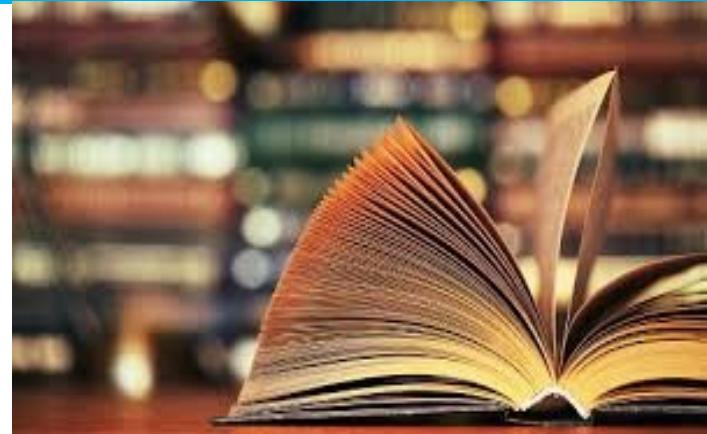
Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies is difficult. *IEEE transactions on neural networks*, 5(2) :157–166.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8) :1735–1780.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural network. In *International Conference on Machine Learning (ICML)*.

# Conseil de lectures

## Livres et articles



Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088) :533–536.

Sutskever, H., Vinyals, O., Le, Q. L. (2014). Sequence to Sequence Learning with Neural Networks, in *International Conference on Neural Information Processing Systems (NIPS)*

Vaswani, A et al. (2017) Attention is All you Need, in *International Conference on Neural Information Processing Systems (NIPS)*.

# Conseil de lectures

Quelques blogs sur les RNN/LSTM (et autr

Blog de [Christopher Olah](#)

Post de [Charles Crousepeyre](#) sur medium.com  
(retranscription d'un tutorial vidéo de Facebook AI)

Post de [NVIDIA](#) sur la traduction automatique

Post de [Michael Nguyen](#) sur towardsdatascience.com



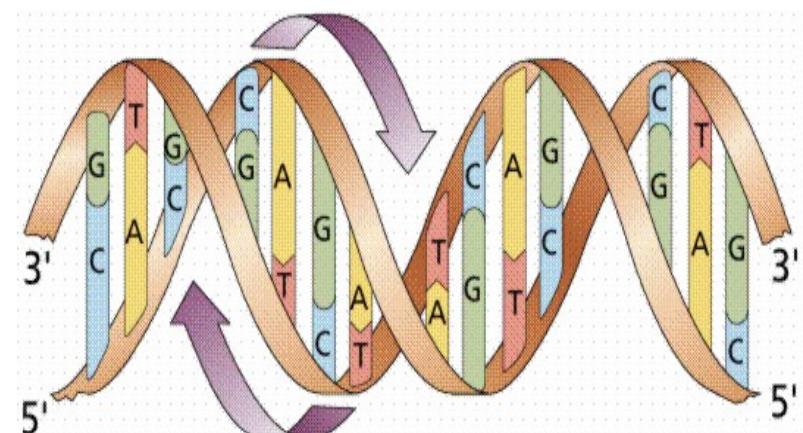
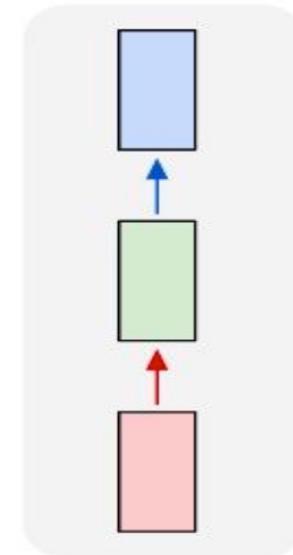
1.

Introduction :  
Pourquoi modéliser des  
séquences ?

# Séquence & mémoire

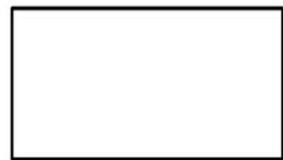
- Préliminaire
  - Tâche de classification « one-to-one » / instantanée
  - Hypothèse : les données suivent une loi I.I.D (identiquement et indépendamment distribuées)
- Des données organisées
  - Espace : cohérence spatiale
  - Temps (ou espace 1-D) : Ordonnancement/séquencement
  - Besoin de « mémoire » du passé
- Exemples de données séquentielles
  - texte, séquence ADN, variations de température, cours de la bourse

**one to one**



# Un exemple

I am Bond , James



McGuire

Bond

tired

am

!

# Un exemple

I am Bond , James

Bond

McGuire

Bond

tired

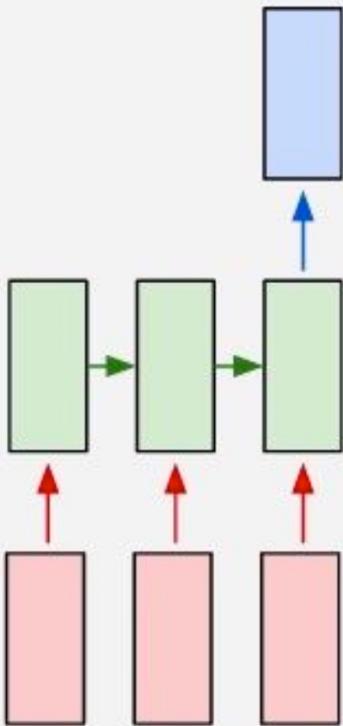
am

!

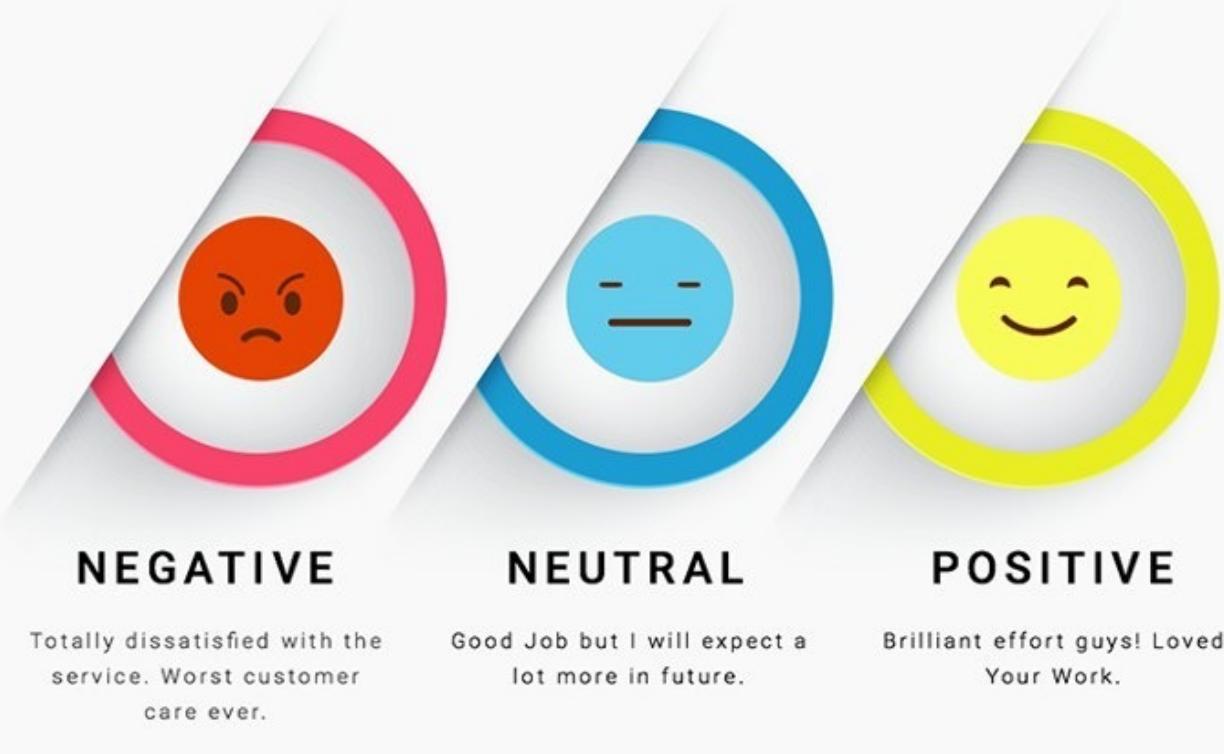


# Architectures et applications

many to one

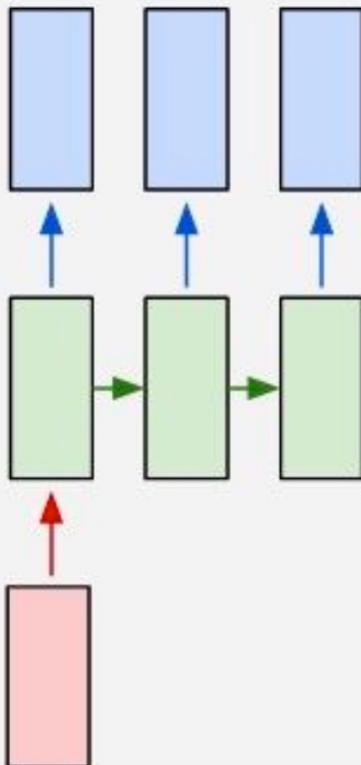


## SENTIMENT ANALYSIS



# Architectures et applications

one to many



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



A dog is running in the grass with a frisbee



A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



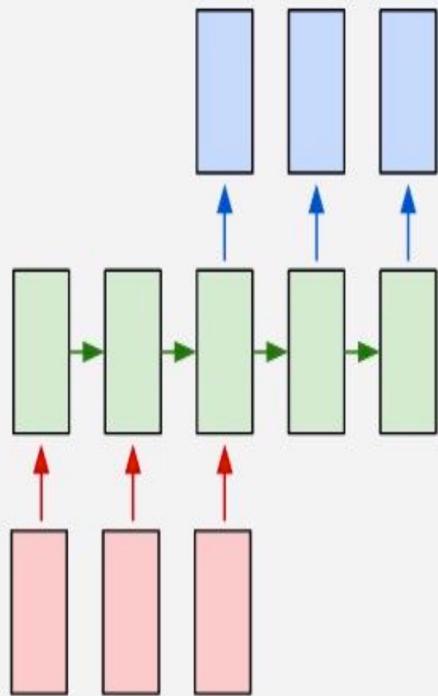
Two giraffes standing in a grassy field



A man riding a dirt bike on a dirt track

# Architectures et applications

many to many

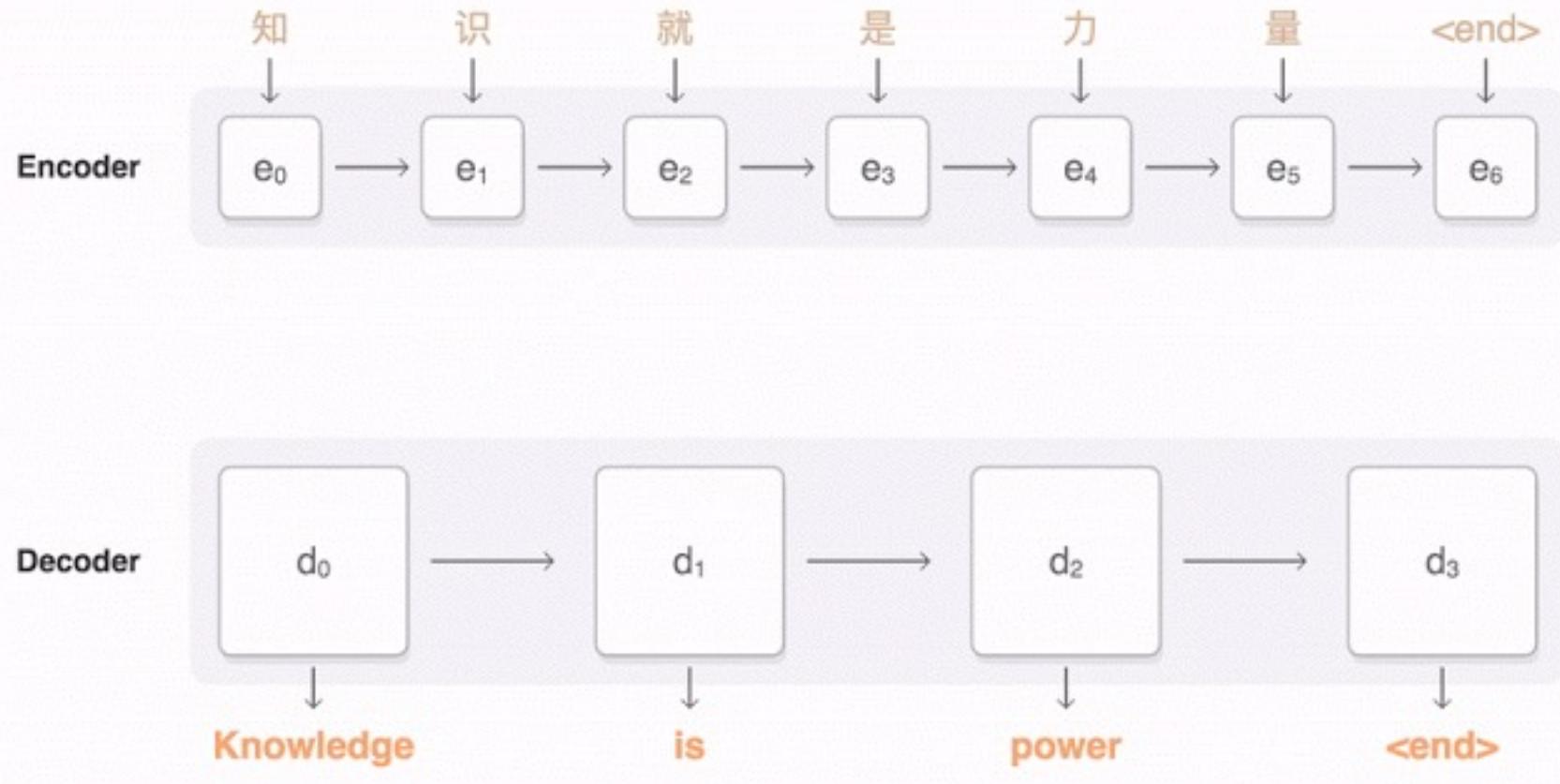


Machine translation



# Architectures et applications

Machine translation : modèles séquence-à-séquence avec mécanisme d'attention



2.

Réseaux de neurones  
récurrents :  
Théorie

# Formalisation

- On souhaite modéliser la séquence,

$$p(\mathbf{y}) = p(y_1, \dots, y_T)$$

- En appliquant Bayes une fois,

$$p(\mathbf{y}) = p(y_1, \dots, y_{T-1}) p(y_T | y_1, \dots, y_{T-1})$$

- En appliquant Bayes de proche en proche,

$$p(\mathbf{y}) = \prod_{t=1}^T p(y_t | y_1, \dots, y_{t-1})$$

- Soit,

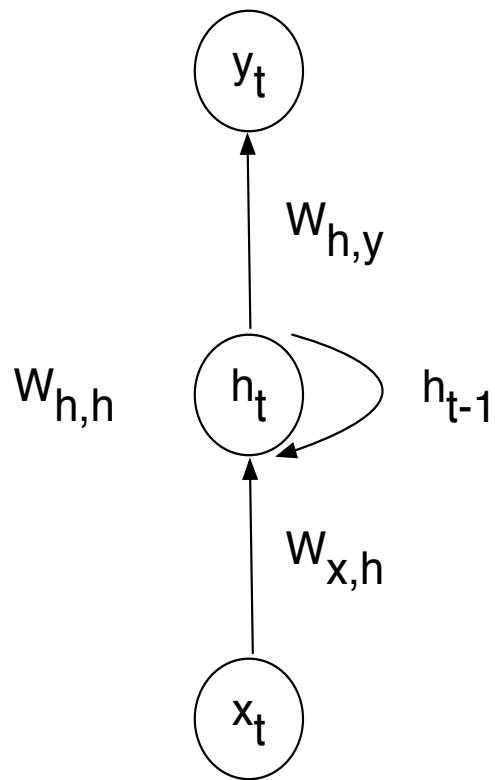
$$p(\mathbf{y}) = \prod_{t=1}^T p(y_t | y_{<t})$$

# Définition

Sortie

Mémoire

Entrée



Nouveauté:  
mémoire du passé

Le passé est  
représenté par la  
récursion sur les  $h_t$   
(boucle de récursion)

Définition :

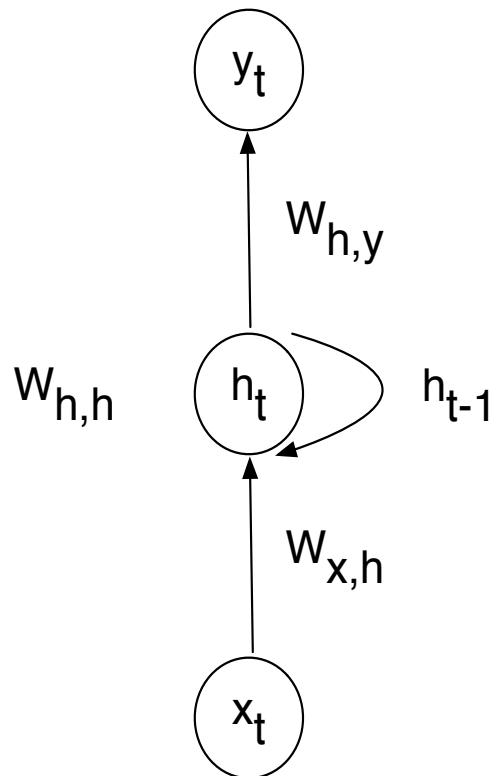
$$h_t = \tanh(W_{x,h} x_t + W_{h,h} h_{t-1})$$
$$y_t = \text{softmax}(W_{h,y} h_t)$$

# Définition

Sortie

Mémoire

Entrée



- Ainsi la probabilité,

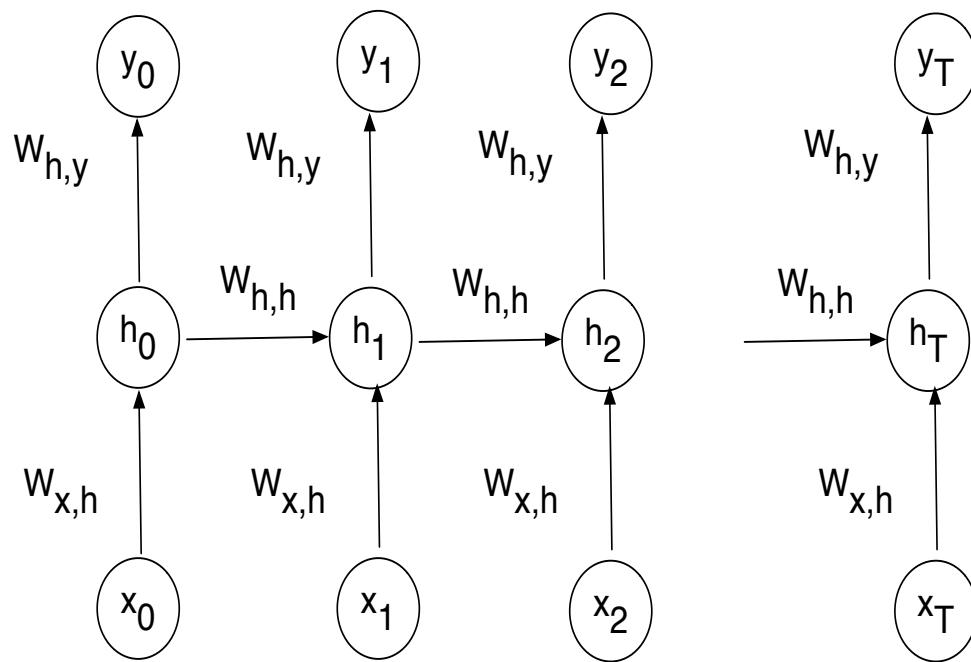
$$p(\mathbf{y}) = \prod_{t=1}^T p(y_t | y_{<t})$$

- S'écrit sous forme RNN avec,

$$p(y_t | y_{<t}) = g(y_{t-1}, h_t)$$

# Graphé déplié

Sortie

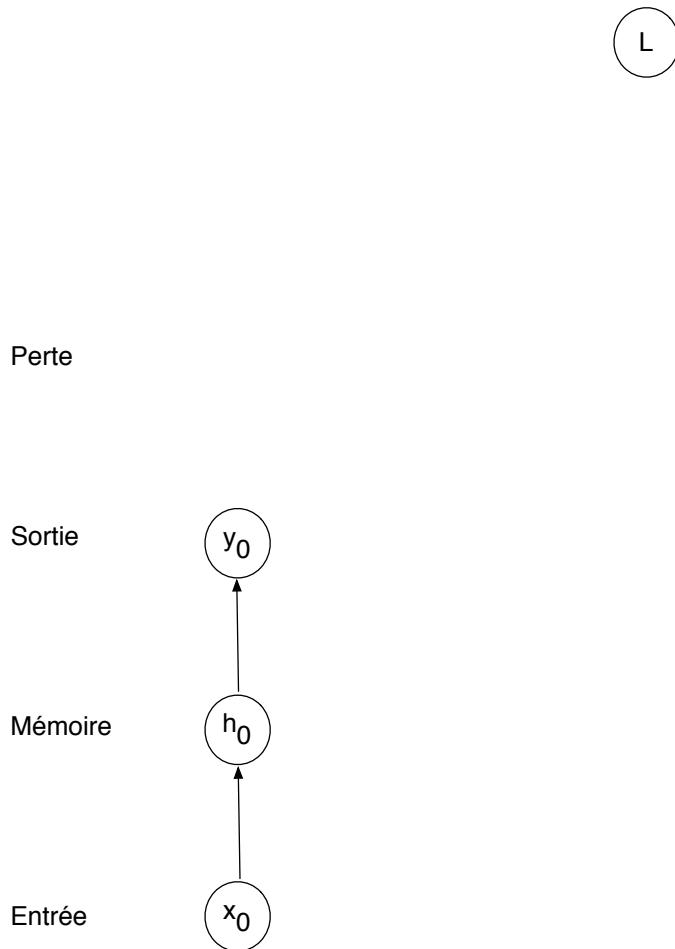


Stationnarité  
des paramètres

$W_{x,h}, W_{h,h}$ , et  
 $W_{h,y}$  sont  
partagés dans  
le temps

Un réseau  
« classique »

# Propagation-avant et calcul d'erreur

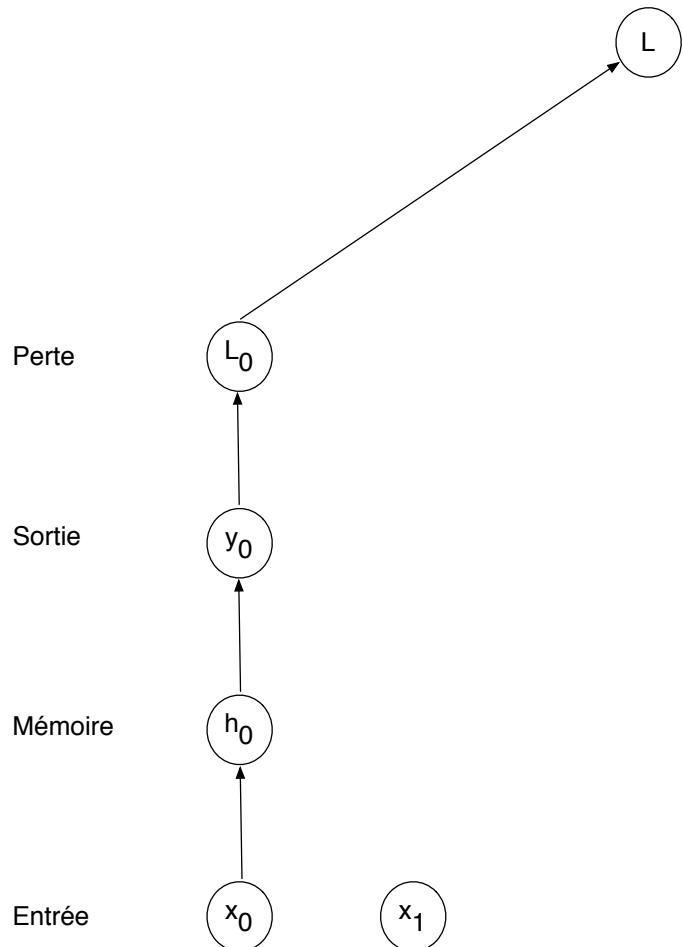


A l'instant 0 :

- On propage l'entrée dans la mémoire
- On déduit la sortie

Perte

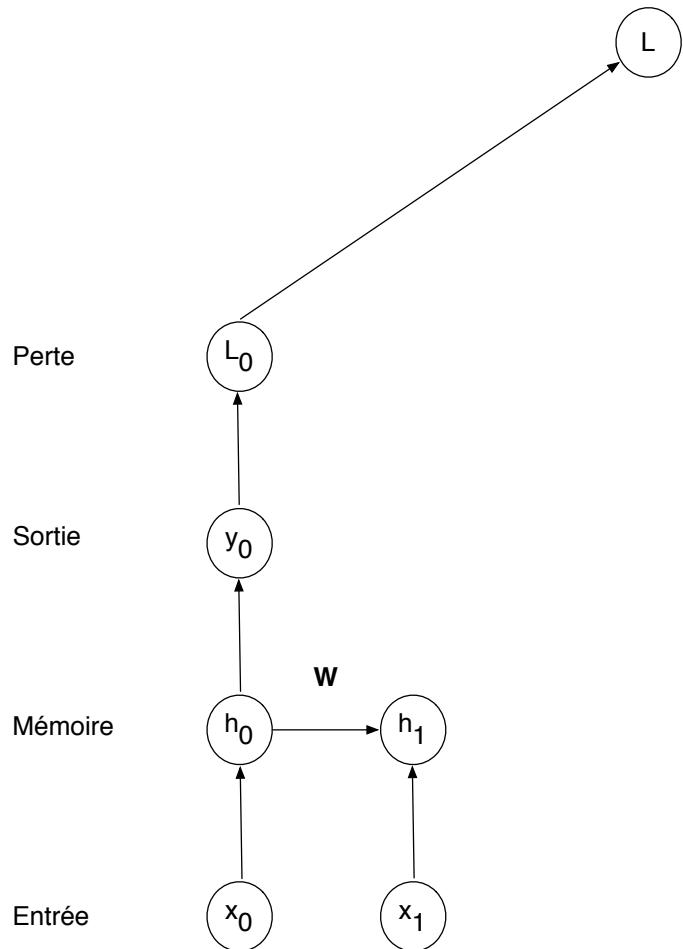
# Propagation-avant et calcul d'erreur



A l'instant 0 :

- On propage l'entrée dans la mémoire
- On déduit la sortie
- On calcule la perte partielle à l'instant 0

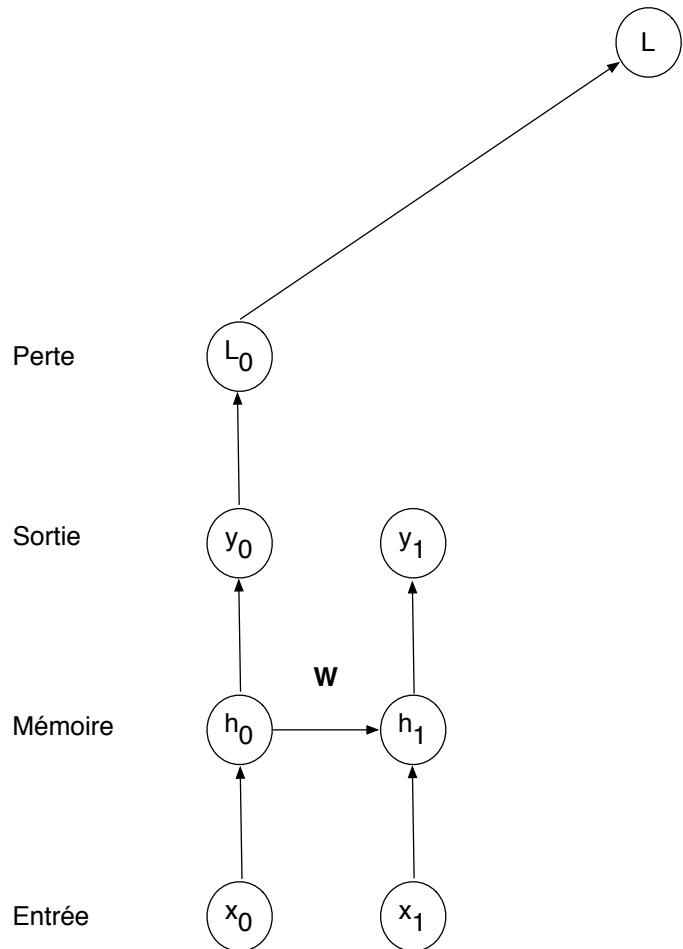
# Propagation-avant et calcul d'erreur



A l'instant 1 :

- On calcul la mémoire à partir de l'entrée  $x_1$  et de l'état de la mémoire précédente

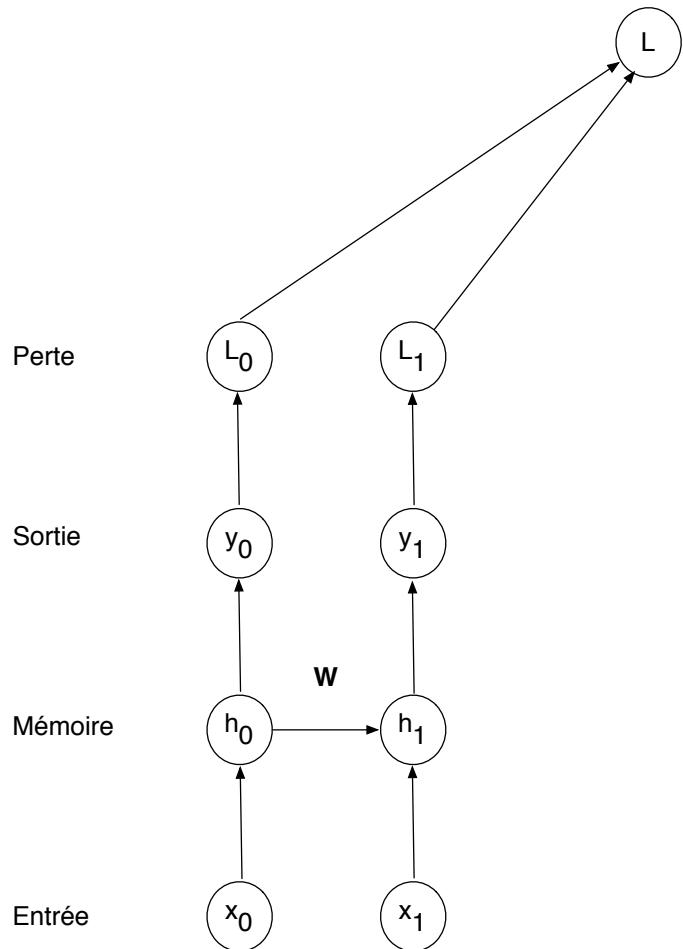
# Propagation-avant et calcul d'erreur



A l'instant 1 :

- On calcule la mémoire à partir de l'entrée  $x_1$  et de l'état de la mémoire précédente
- On déduit la sortie

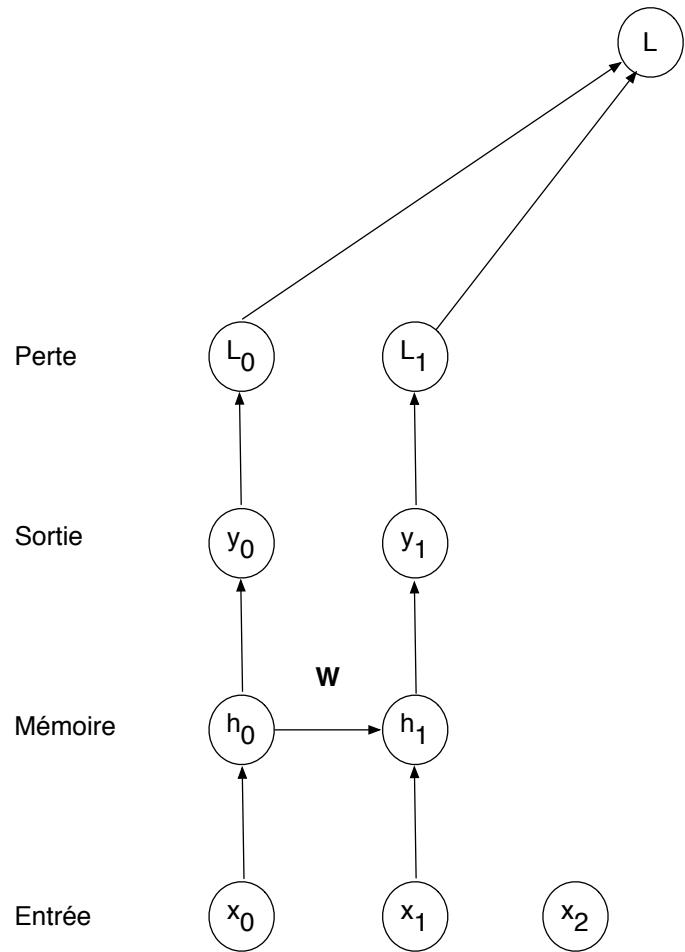
# Propagation-avant et calcul d'erreur



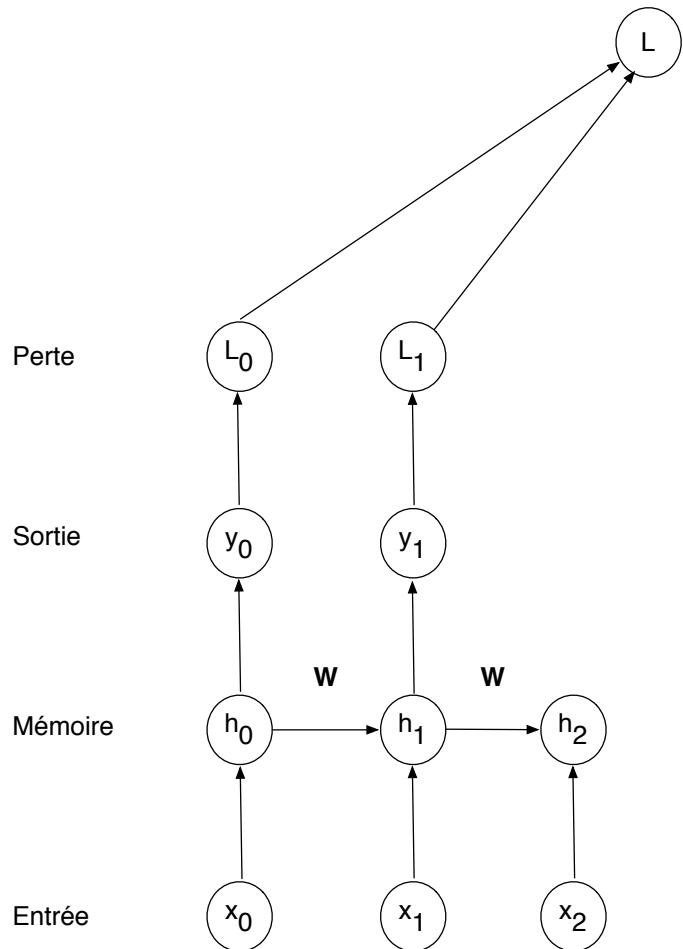
A l'instant 1 :

- On calcule la mémoire à partir de l'entrée  $x_1$  et de l'état de la mémoire précédente
- On déduit la sortie
- On calcule la perte partielle à l'instant 1
- On cumule pour le calcul de la perte totale

# Propagation-avant et calcul d'erreur



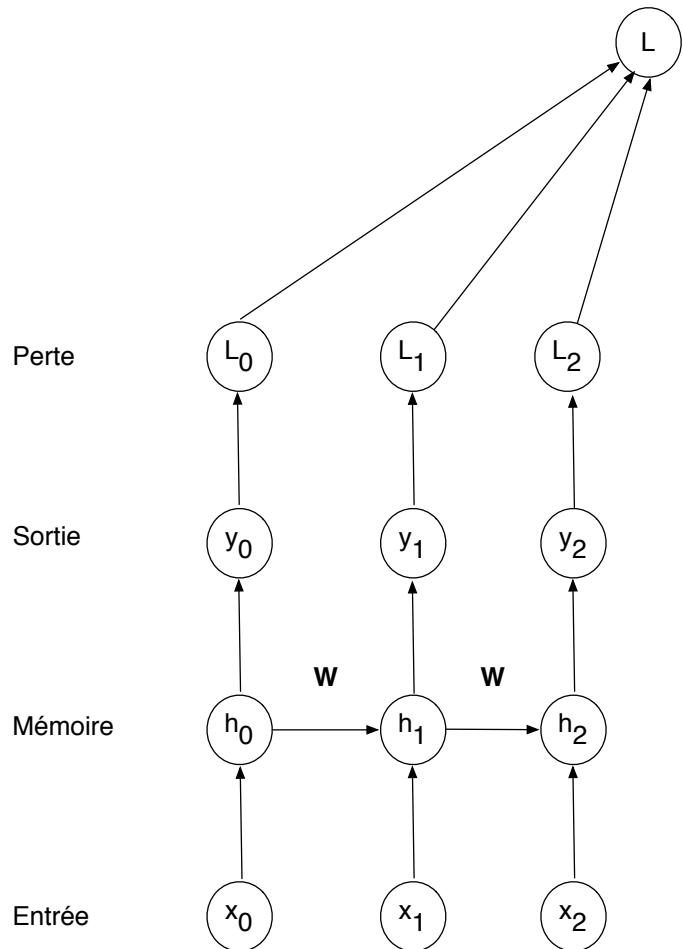
# Propagation-avant et calcul d'erreur



A l'instant  $t=2$  :

- On calcul la mémoire  $h_2$  à partir de l'entrée  $x_2$  et de l'état de la mémoire précédente  $h_1$

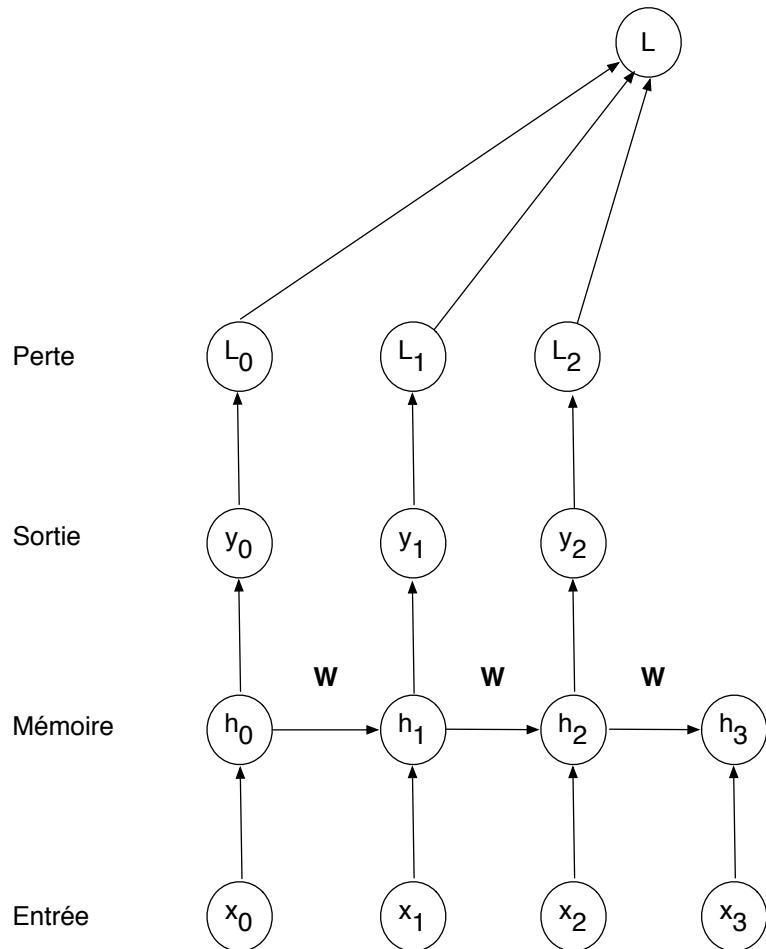
# Propagation-avant et calcul d'erreur



A l'instant  $t=2$  :

- On calcule la mémoire  $h_2$  à partir de l'entrée  $x_2$  et de l'état de la mémoire précédente  $h_1$
- On déduit la sortie  $y_2$
- On calcule la perte partielle  $L_2$  à l'instant 0

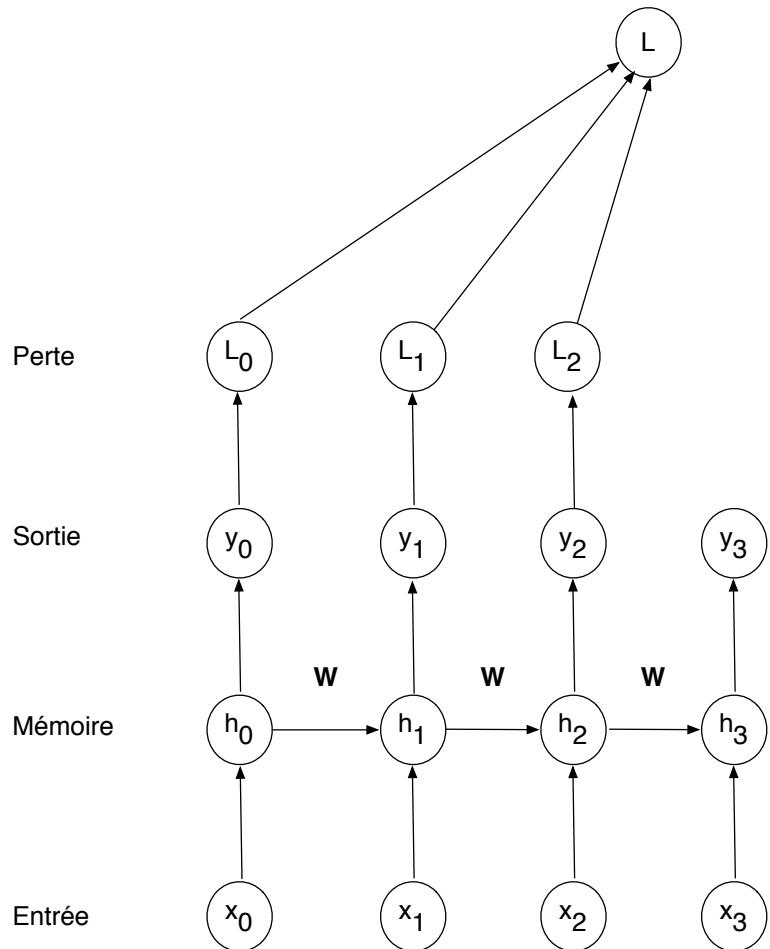
# Propagation-avant et calcul d'erreur



A l'instant  $t=3$  :

- On calcul la mémoire  $h_3$  à partir de l'entrée  $x_3$  et de l'état de la mémoire précédente  $h_2$
- On déduit la sortie  $y_3$
- On calcule la perte partielle  $L_3$

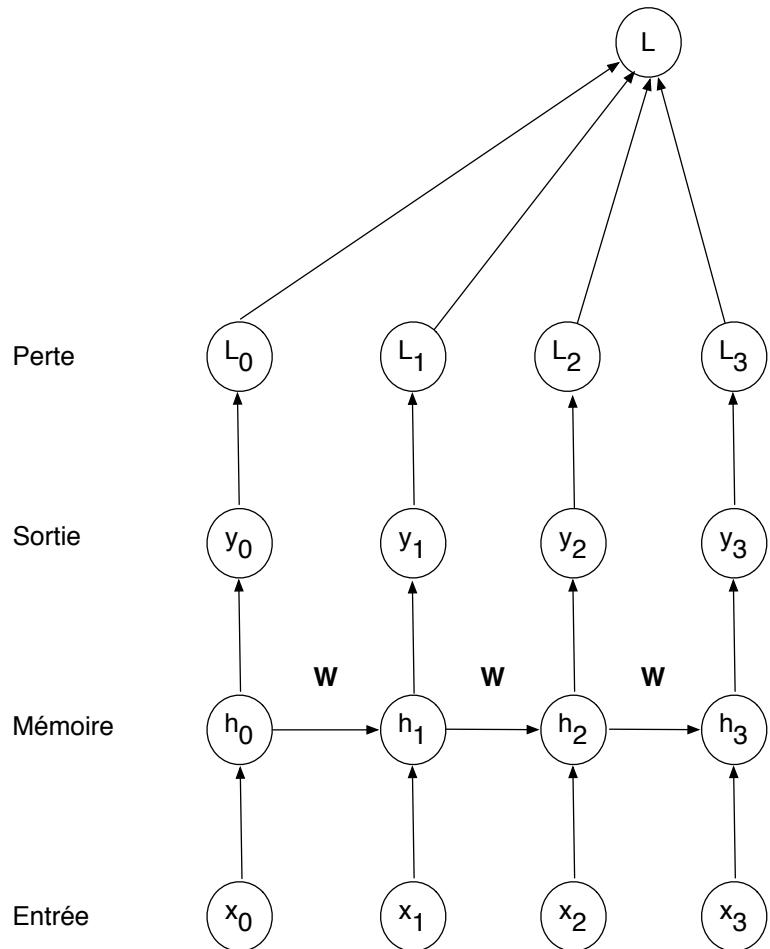
# Propagation-avant et calcul d'erreur



A l'instant  $t=3$  :

- On calcul la mémoire  $h_3$  à partir de l'entrée  $x_3$  et de l'état de la mémoire précédente  $h_2$
- On déduit la sortie  $y_3$
- On calcule la perte partielle  $L_3$

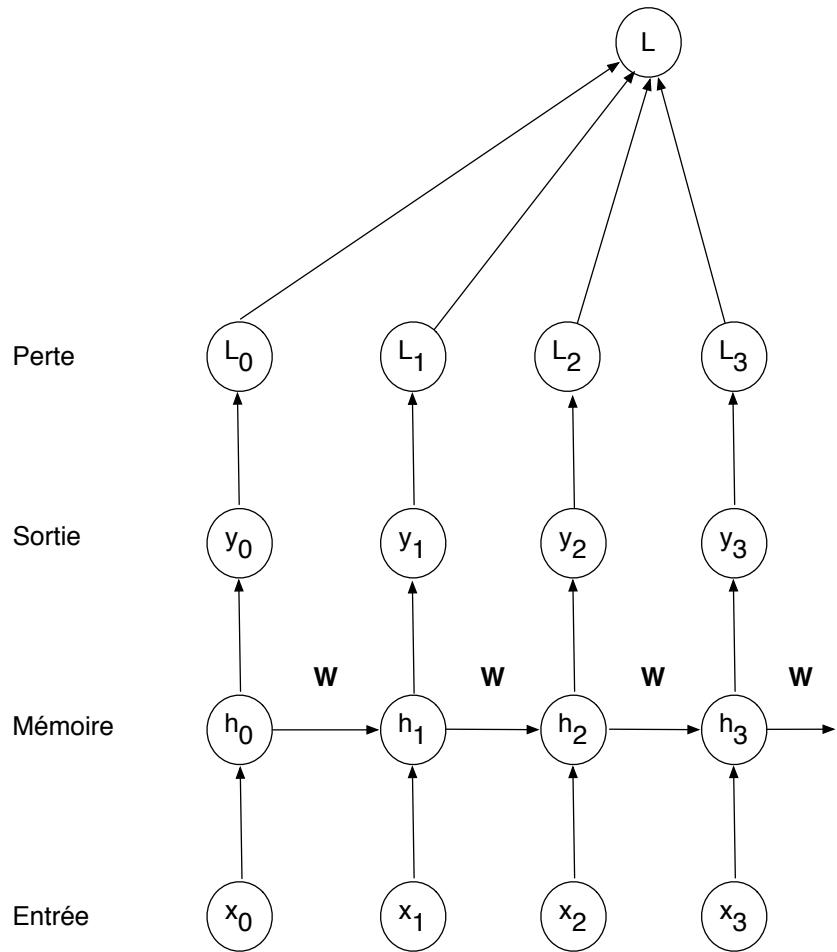
# Propagation-avant et calcul d'erreur



A l'instant  $t=3$  :

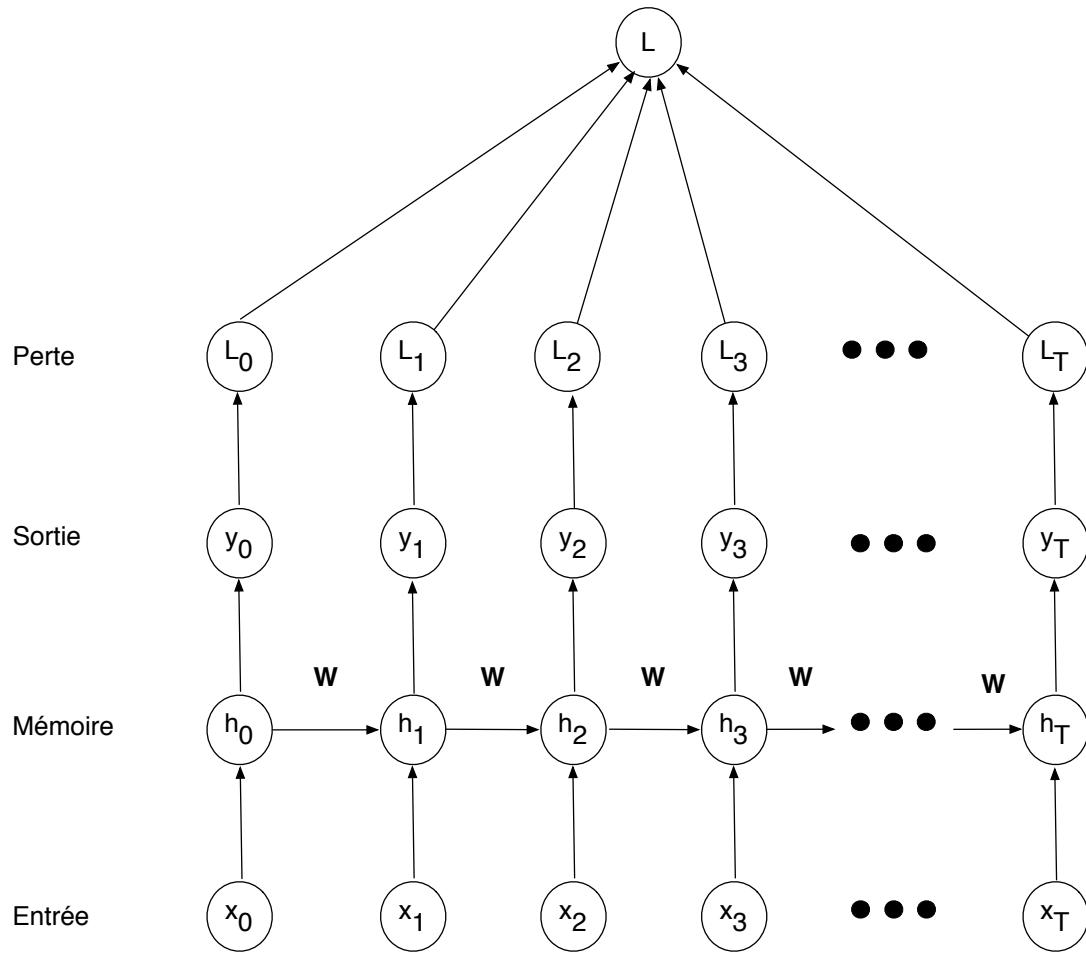
- On calcule la mémoire  $h_3$  à partir de l'entrée  $x_3$  et de l'état de la mémoire précédente  $h_2$
- On déduit la sortie  $y_3$
- On calcule la perte partielle  $L_3$
- On cumule dans la perte totale  $L$

# Propagation-avant et calcul d'erreur



On ré-itère jusqu'à la fin de la séquence !

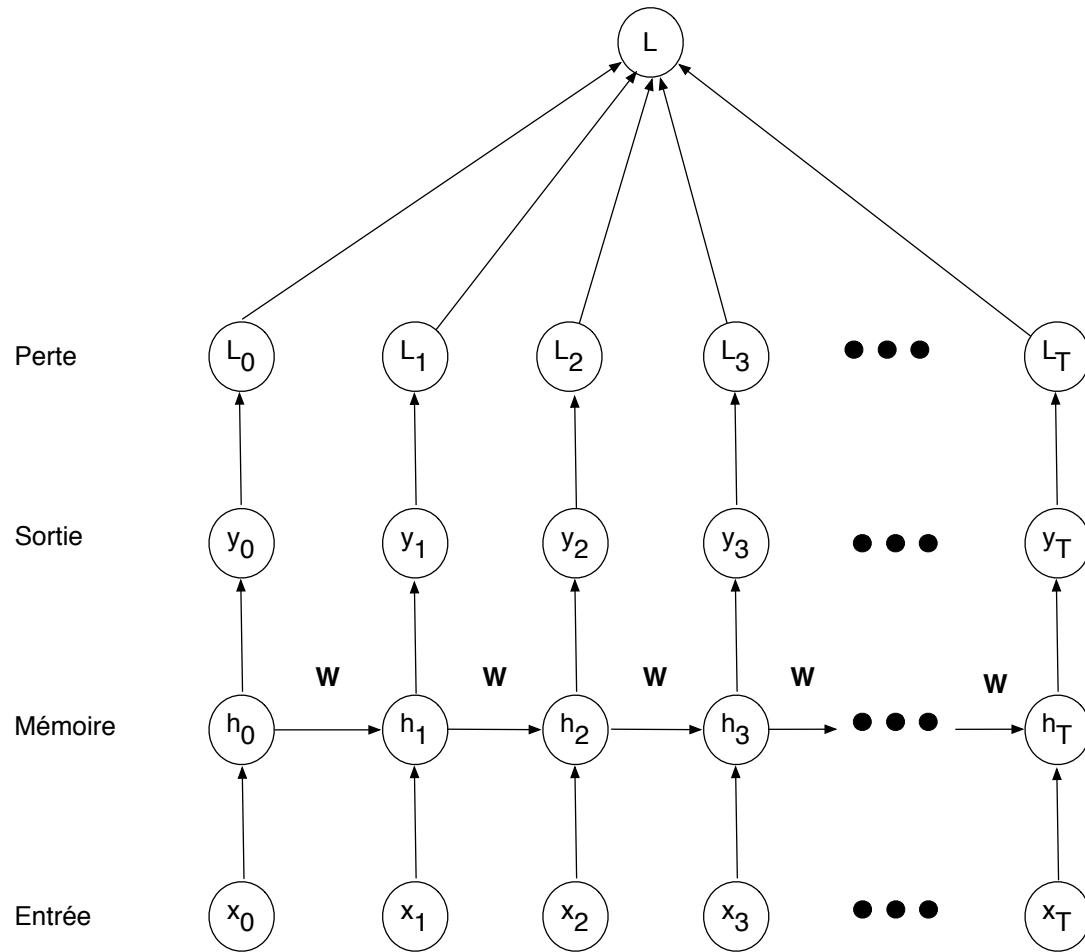
# Propagation-avant et calcul d'erreur



On ré-itère jusqu'à la fin de la séquence !

$$\begin{aligned} & y = f(x) + x^2 \\ & \frac{dy}{dx} = 2x \\ & \int_{t_0}^t dt = \int_{t_0}^t dx \\ & f(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h} \end{aligned}$$

# Rétro-propagation dans le temps (BPTT)



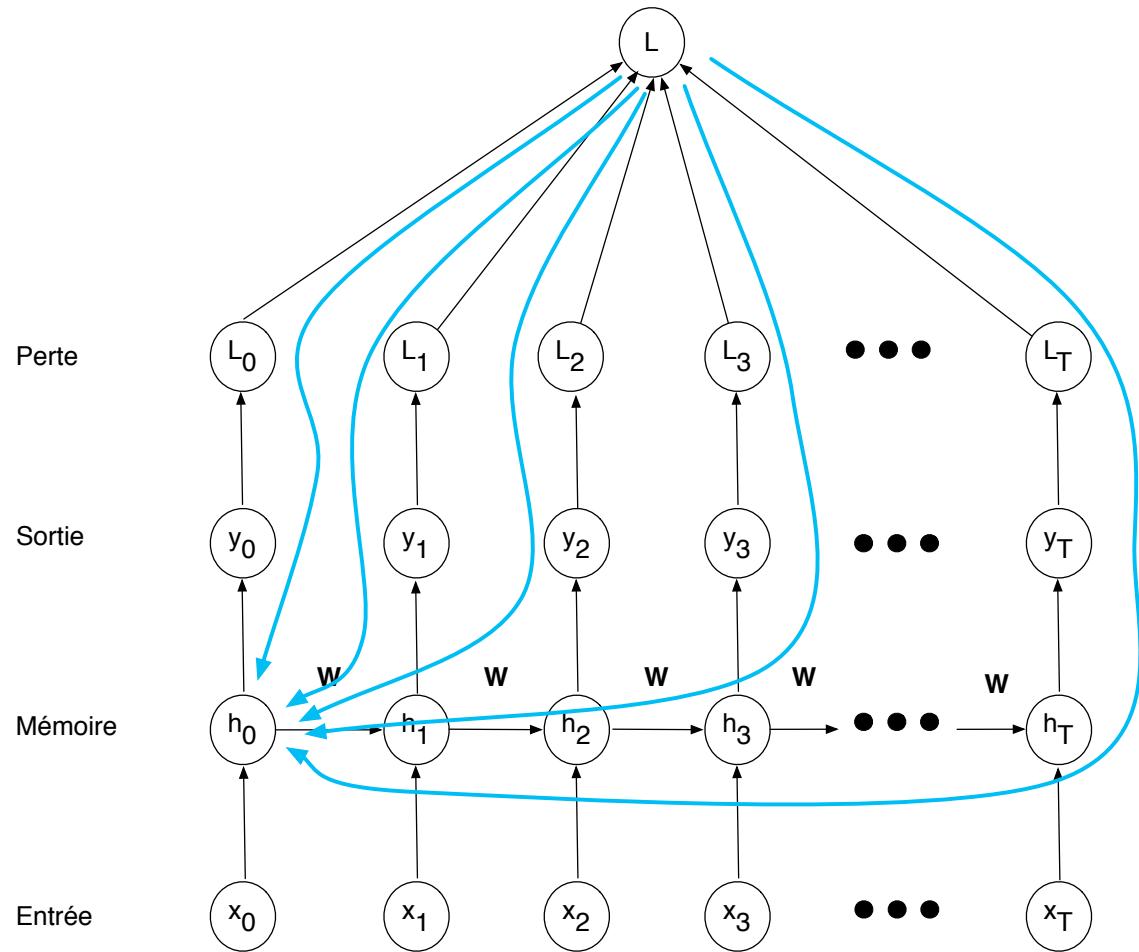
La rétro-propagation est calculée classiquement par descente de gradient :

$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

On doit donc calculer :

$$\frac{\partial L}{\partial \mathbf{W}}$$

# Rétro-propagation dans le temps (BPTT)

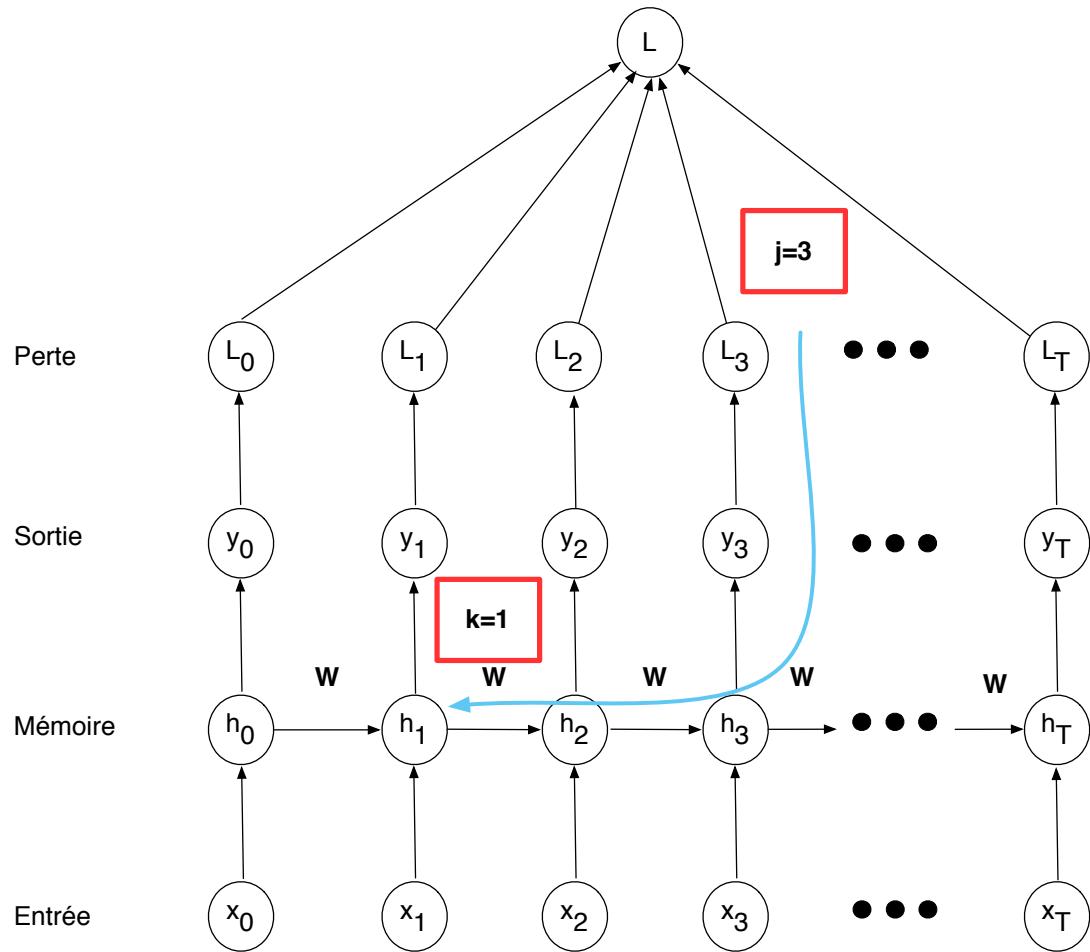


Une contribution cumulée

Il faut passer par tous les chemins qui partent de  $L$  et remontent à l'origine :

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^T \frac{\partial L_j}{\partial \mathbf{W}}$$

# Rétro-propagation dans le temps (BPTT)

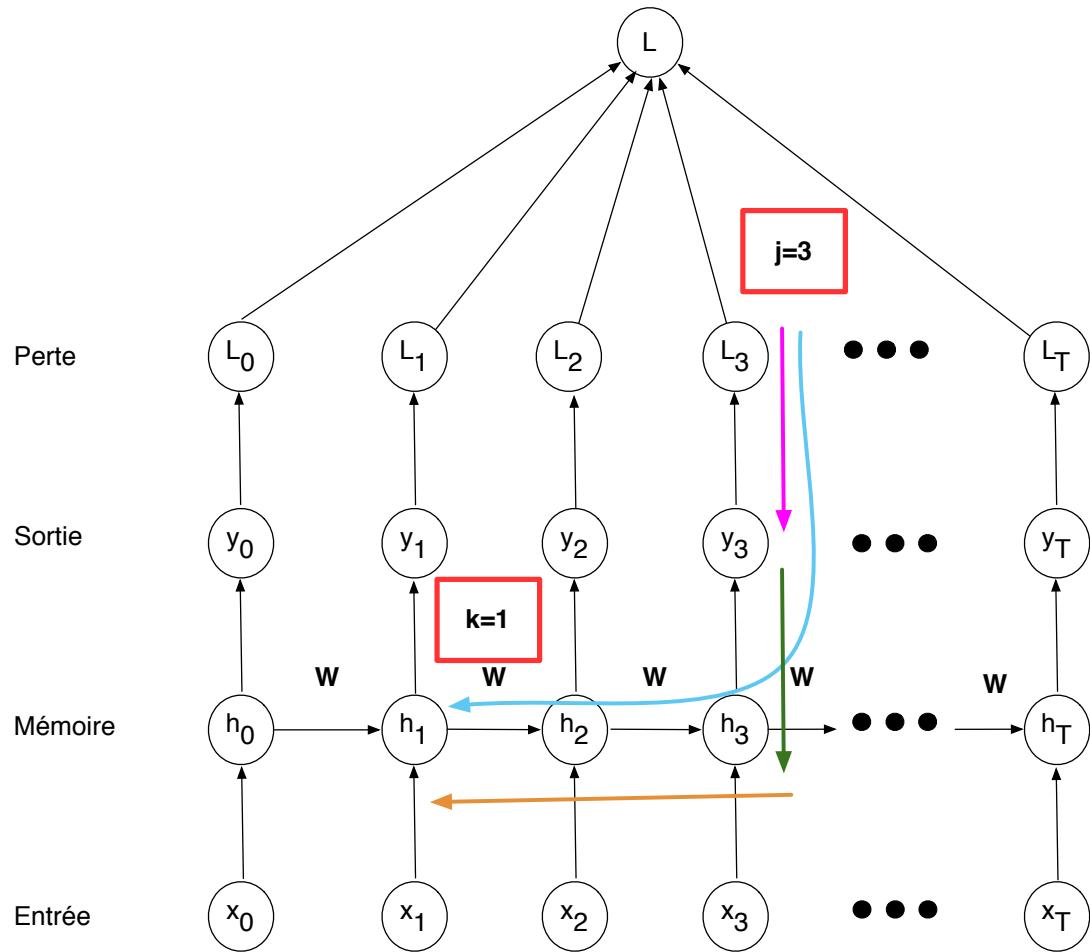


## 2. Règle de la chaîne (1/3):

Et chaque  $L$  à l'instant  $j$  dépend de toutes les matrices de poids aux instants  $k$  du passé :

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^j \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

# Rétro-propagation dans le temps (BPTT)



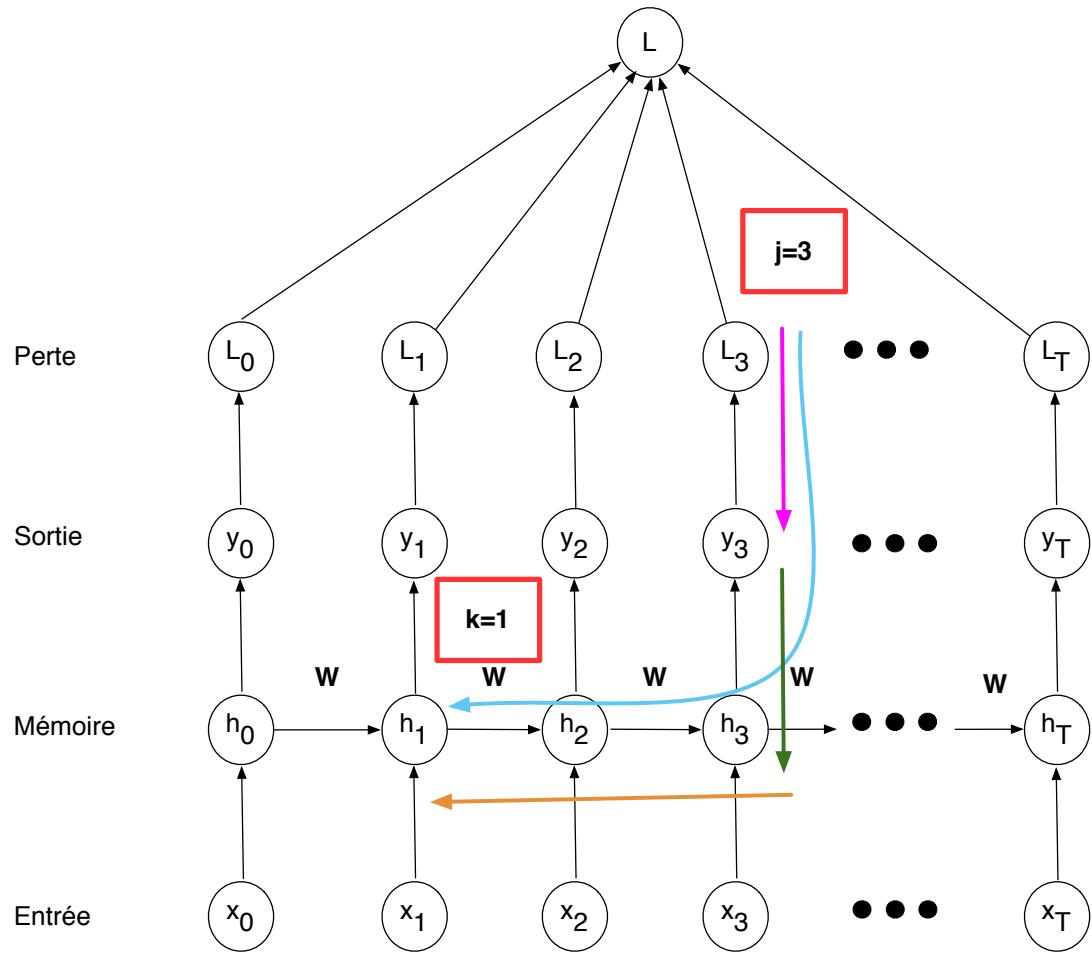
## 3. Règle de la chaîne (2/3):

La dépendance des  $L_j$  sur les  $h_k$  n'est pas explicite

On applique la règle en chaîne pour décomposer le calcul :

$$\frac{\partial L_j}{\partial h_k} = \boxed{\frac{\partial L_j}{\partial y_j}} \quad \boxed{\frac{\partial y_j}{\partial h_j}} \quad \boxed{\frac{\partial h_j}{\partial h_k}}$$

# Rétro-propagation dans le temps (BPTT)



## 3. Règle de la chaîne (2/3):

La dépendance des  $L_j$  sur les  $h_k$  n'est pas explicite

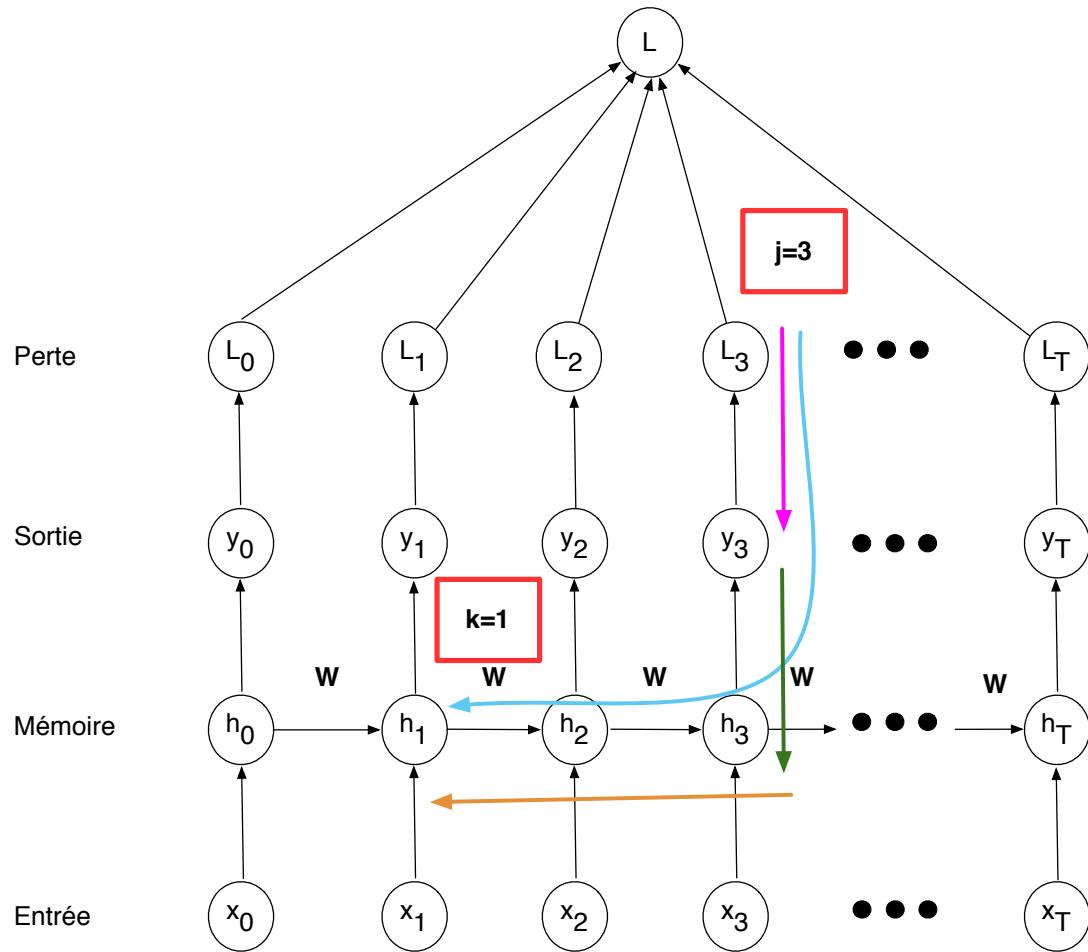
On applique la règle en chaîne pour décomposer le calcul :

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$



Dérivations déjà vues  
pour le MLP

# Rétro-propagation dans le temps (BPTT)



4. Règle de la chaîne  
(3/3):

Idem pour la dépendance entre  $h_j$  et  $h_k$

On applique la règle de proche en proche :

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}}$$

# Rétro-propagation dans le temps (BPTT)

Contribution de  $\mathbf{W}_h$  à l'erreur totale :

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^T \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

On calcule cette quantité sur la séquence entière, puis on opère un pas de descente de gradient

Possibilité de réduire la complexité en tronquant la BPTT

# Dispersion/explosion du gradient

En utilisant l'expression de  $h_m$  dans un RNN:

$$h_m = g(\mathbf{W}_{h,h} h_{m-1} + \mathbf{W}_{x,h} x_m)$$

On peut développer le produit et montrer que :

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^\top \text{diag}(g'(\mathbf{W}_{h,h} h_{m-1} + \mathbf{W}_{x,h} x_m))$$

Une dérivation explicite de proche en proche !

# Dispersion/explosion du gradient

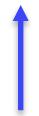
Il peut donc s'exprimer finalement :

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^j \boxed{\mathbf{W}_h^\top} \boxed{\text{diag}(g'(\mathbf{W}_{h,h}h_{m-1} + \mathbf{W}_{x,h}x_m))}$$

Matrice des poids  
du réseau

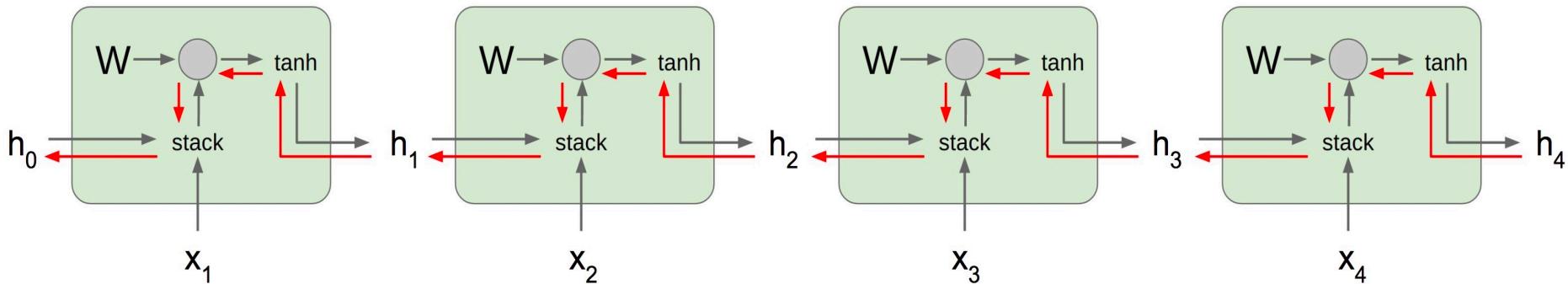


Dérivée de la  
fonction  
d'activation



- La récursion entraîne une dispersion (ou une explosion) exponentielle du gradient
- La correction d'erreur diminue exponentiellement avec l'intervalle entre le passé et le présent
- Mémoire « à court-terme »

# Dispersion/explosion du gradient



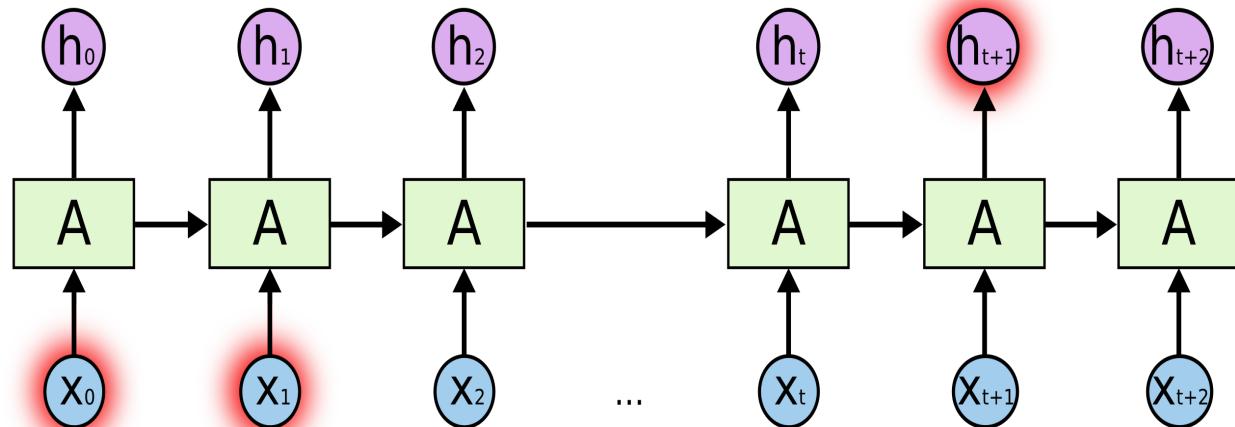
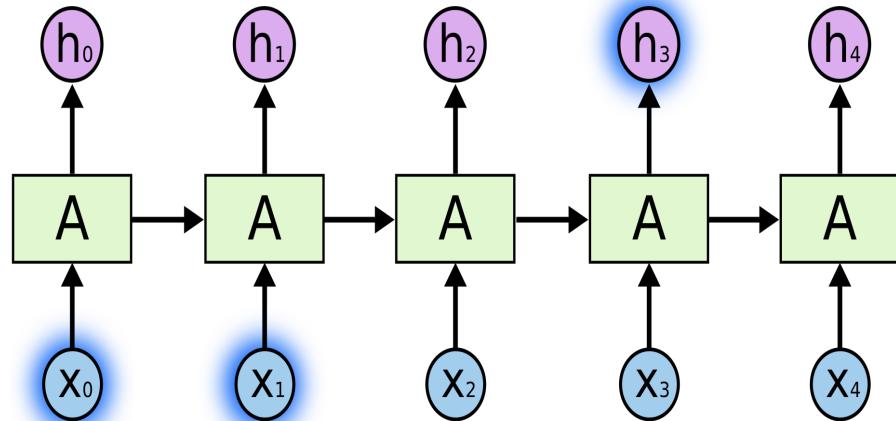
- La récursion entraîne une dispersion (ou une explosion) exponentielle du gradient
- La correction d'erreur diminue exponentiellement avec l'intervalle entre le passé et le présent
- Mémoire « à court-terme »

# Dispersion/explosion du gradient

Mémoire « à court-terme »

Exemple : l  
Le chat boit du lait

Mémoire « à long-  
terme »

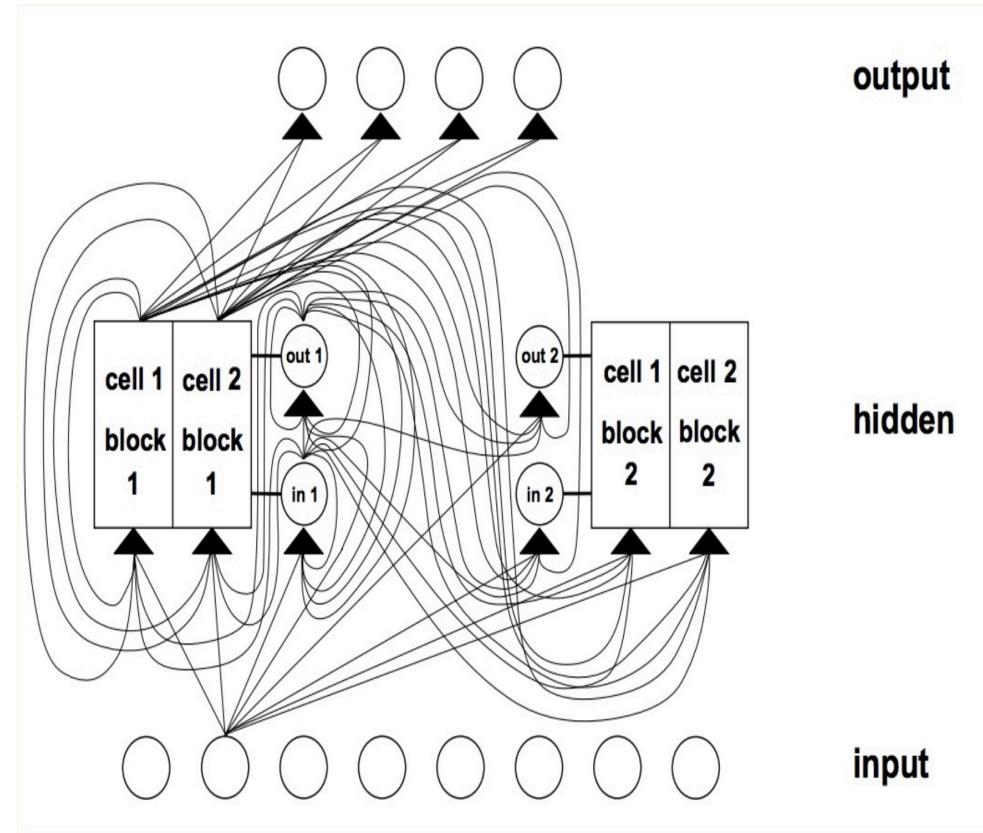


3.

Architectures avancées :  
Ce qui marche vraiment !

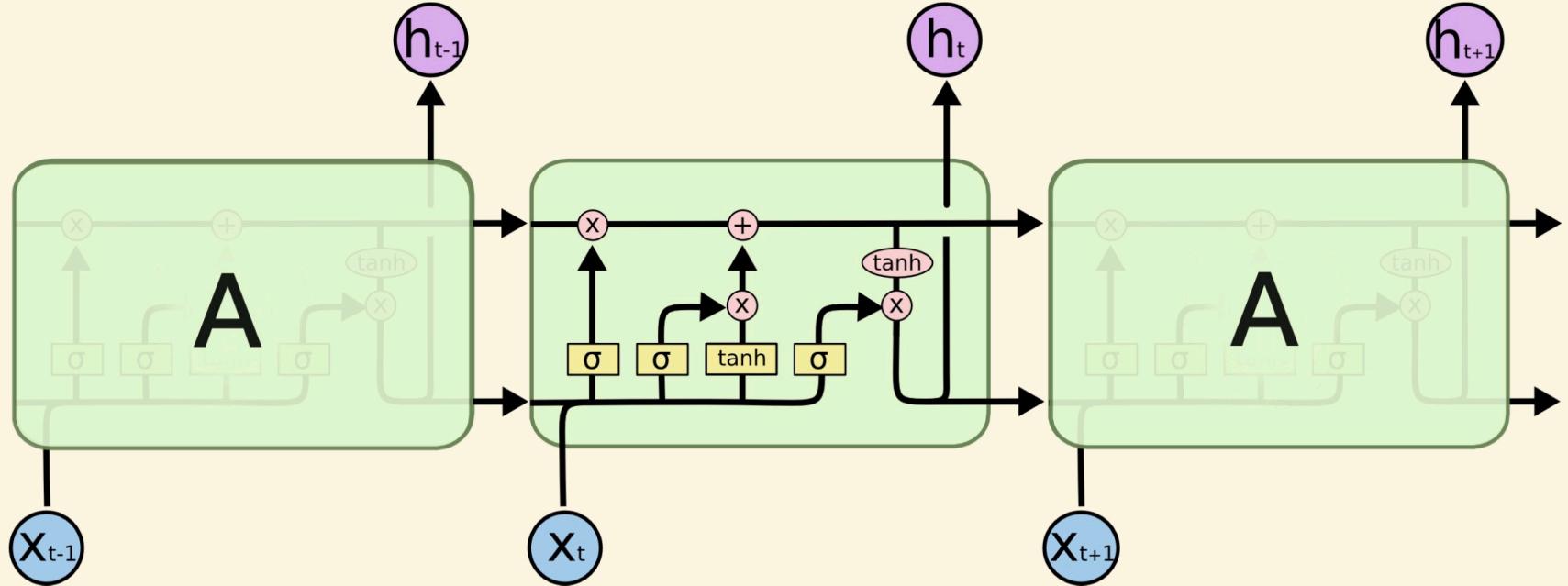
# Les réseaux LSTM

- Il existe des méthodes classiques pour éviter la disparition du gradient : batch-normalization, etc...
- Une avancée décisive est l'invention des réseaux à mémoire court et long terme (LSTM, en anglais)
- L'idée clef est de distinguer : **traitement local de l'information ET transmission globale**



[Hochreiter and Schmidhuber, 1997]

# Représentation graphique



Structure générale similaire à un RNN,  
mais avec comportement interne plus complexe

# Définition

## Vanilla RNN

$$h_t = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

## LSTM

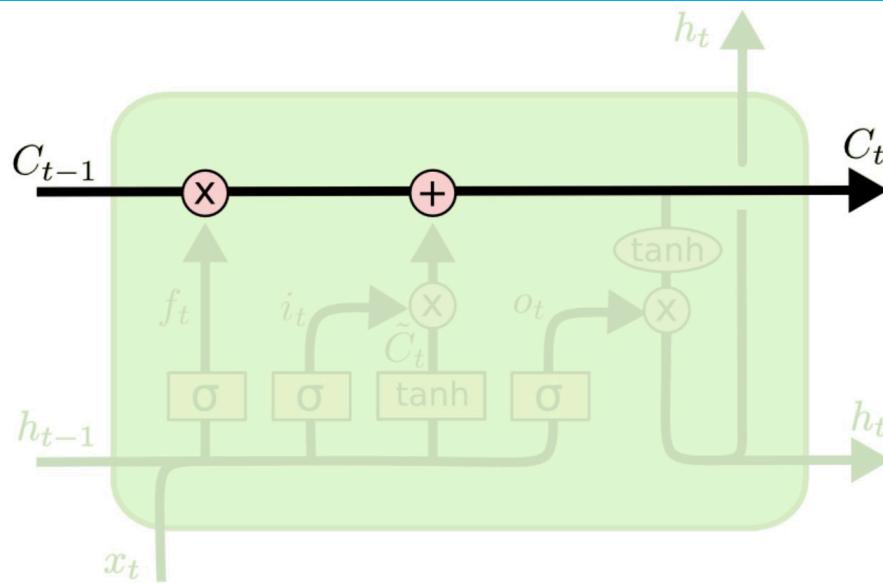
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Structure générale similaire à un RNN,  
mais avec comportement interne plus complexe

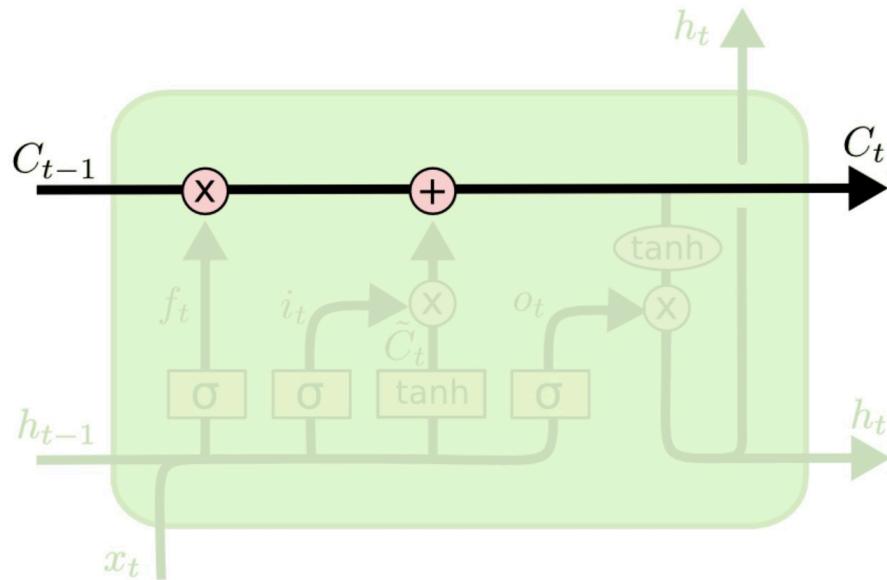
# Idée principale



Une « autoroute » de l'information

Le chemin ne conduit plus à des multiplications répétées sur des constantes : mémoire partagée (mais à court-terme) ou fonction d'activation

# Idée principale

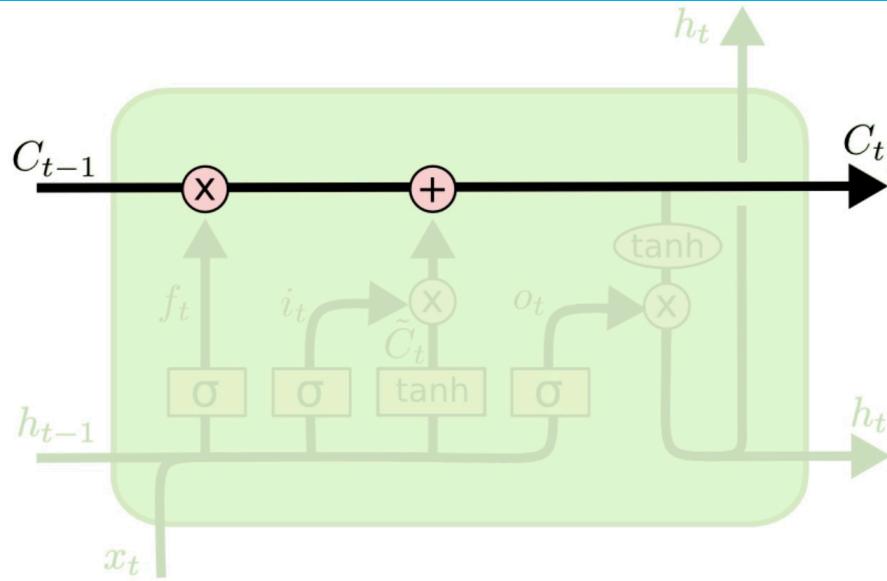


Une « autoroute » de l'information

- L'addition agit comme un « tuyau en T » : combine l'information actuelle avec la mémoire du passé
- Pas de perte ou d'explosion de l'information



# Idée principale



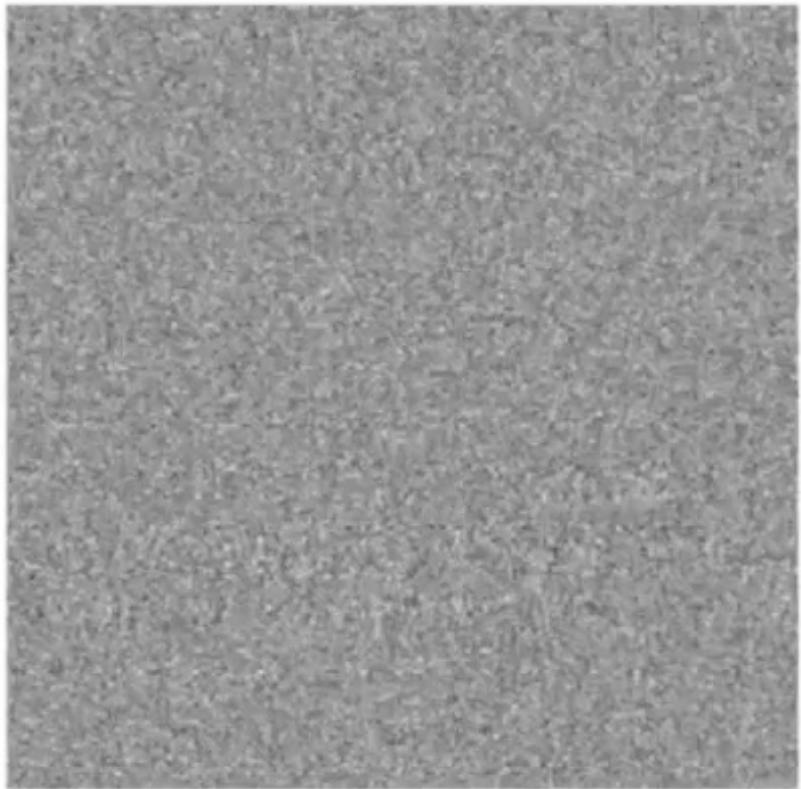
Une « autoroute » de l'information

- La multiplication agit comme une « valve » qui régule la transmission de l'information passée (ce qui reste en mémoire on non)
- Cette régulation est réalisée en interne dans la cellule

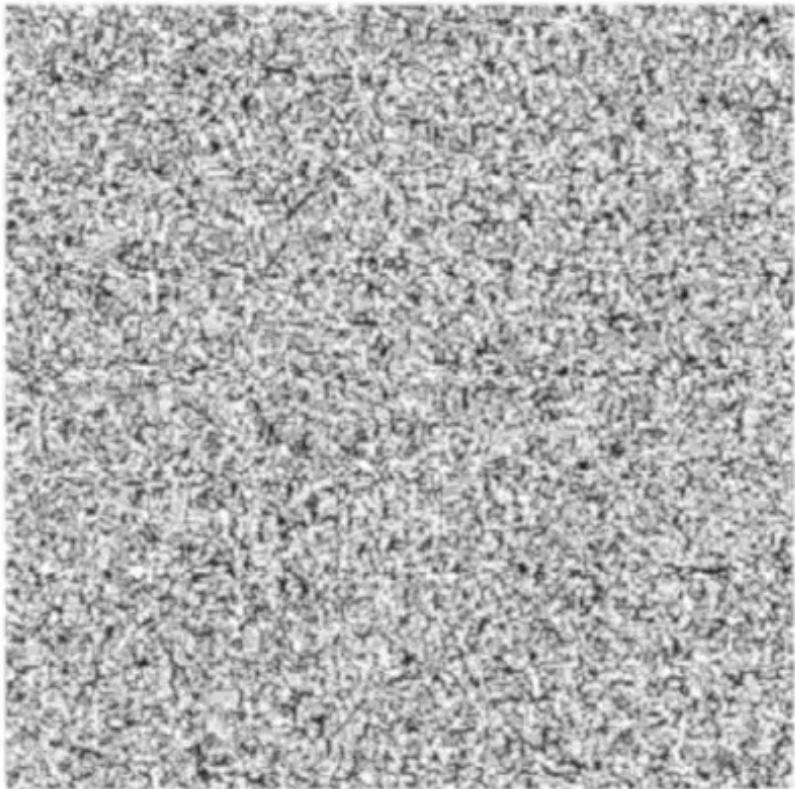


# RNN vs. LSTM : visualisation du gradient

127

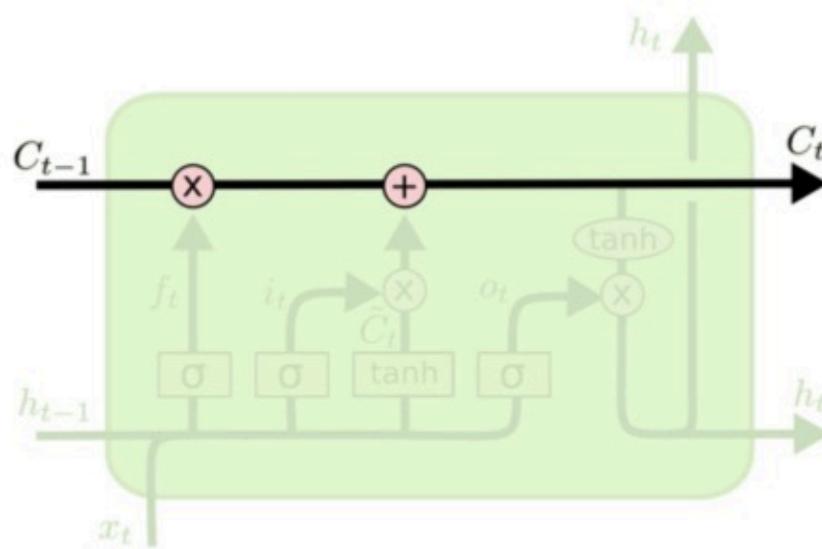


127



# LSTM en détails

Forget what must be forgotten and remember what must be remembered

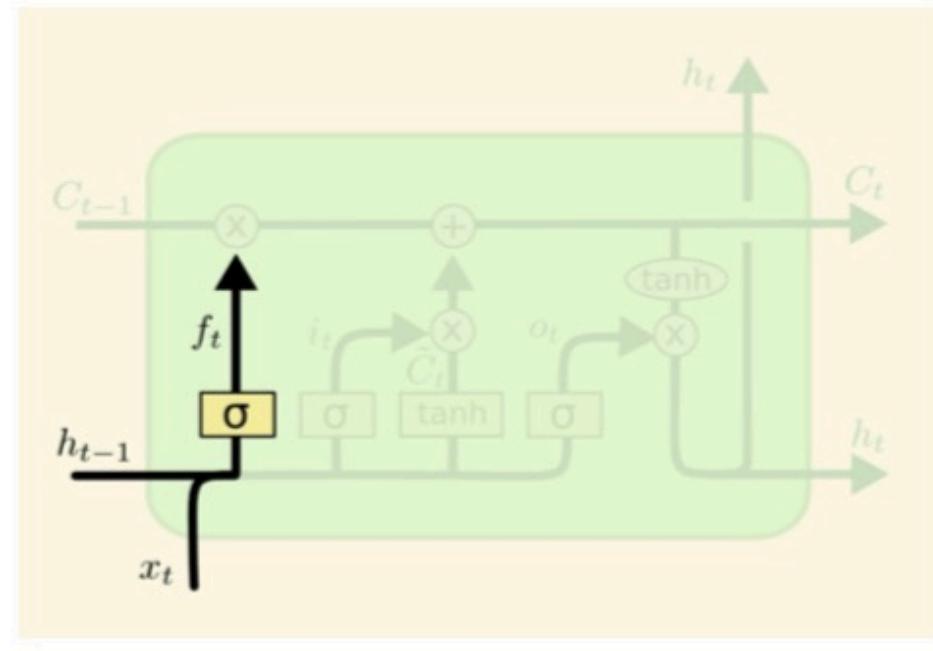


$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad \text{with} \quad \tilde{c}_t = \tanh(W[h_{t-1}, x_t]) \quad (16)$$

- The memory cell can add or delete information from the cell state by **gates**

# LSTM en détails

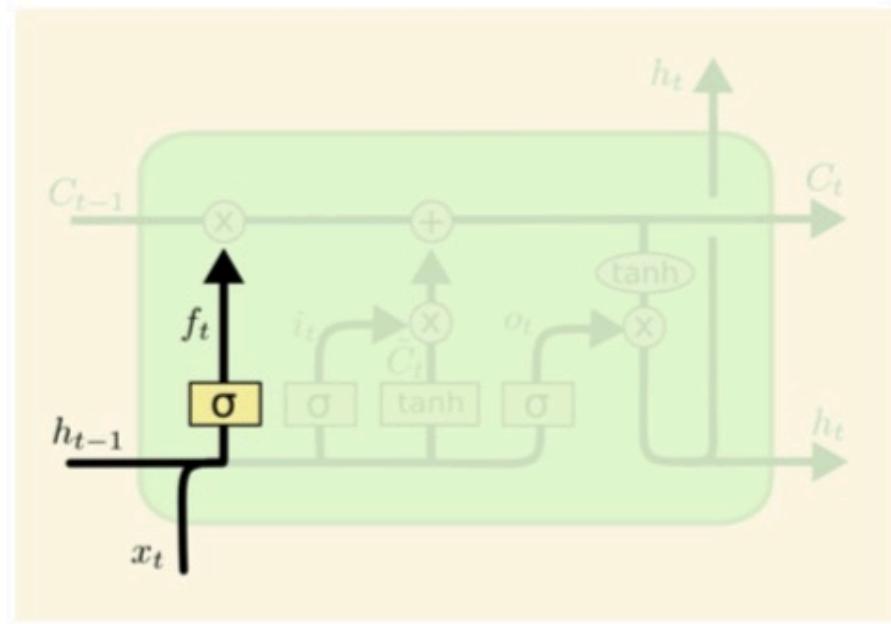
The **forget gate** tells how much of the past information will be forgotten



$$f_t = \sigma(W_f[h_{t-1}, x_t]) \quad (17)$$

# LSTM en détails

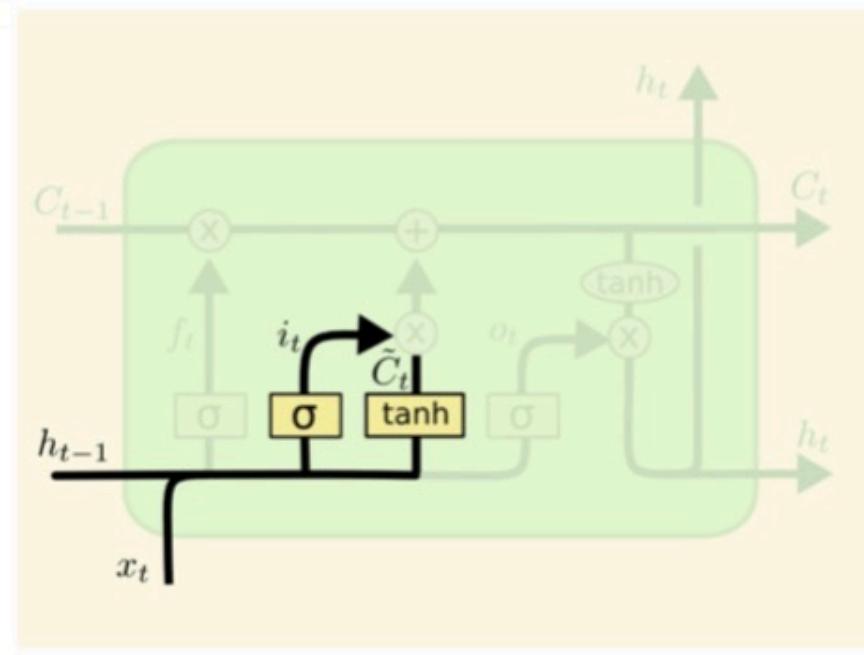
The **forget gate** tells how much of the past information will be forgotten



- ▶ The forget gate is a weight  $f_t$  applied to the previous state  $c_{t-1}$
- ▶  $f_t$  uses previous output  $h_{t-1}$  and current input  $x_t$  and a sigmoid activation function to compute  $f_t$
- ▶ Thanks to the sigmoid,  $f_t \leq 1$ , so the gradient can not explode
- ▶ Then, the information does not vanish if  $f_t \approx 1$ , and is forgotten otherwise

# LSTM en détails

The **input gate** tells how much of the past information is passed to the current cell

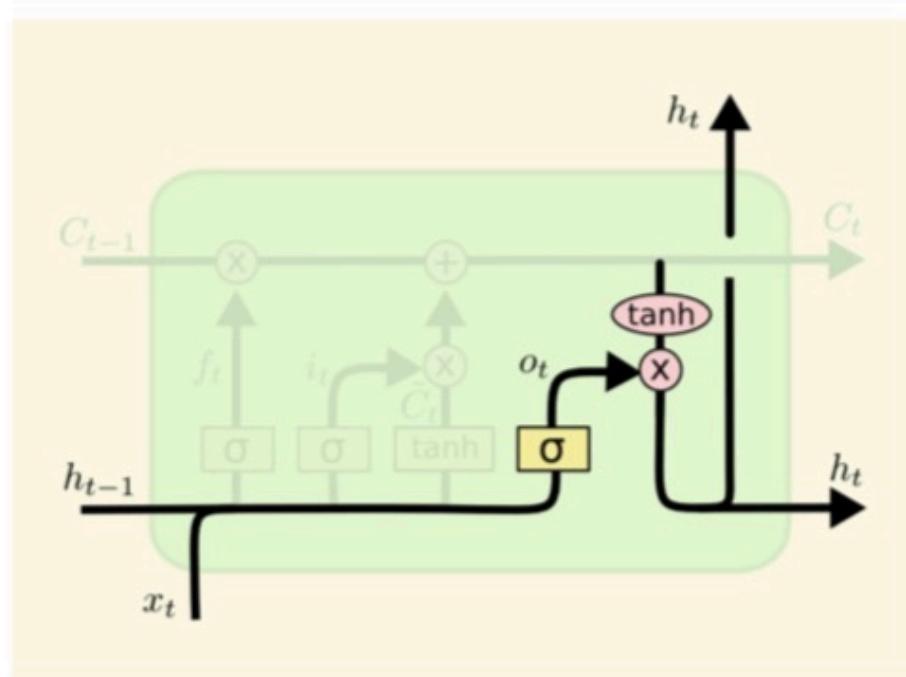


$$i_t = \sigma(W_i[h_{t-1}, x_t]) \quad (18)$$

$$\tilde{c}_t = \tanh([h_{t-1}, x_t]) \quad (19)$$

# LSTM en détails

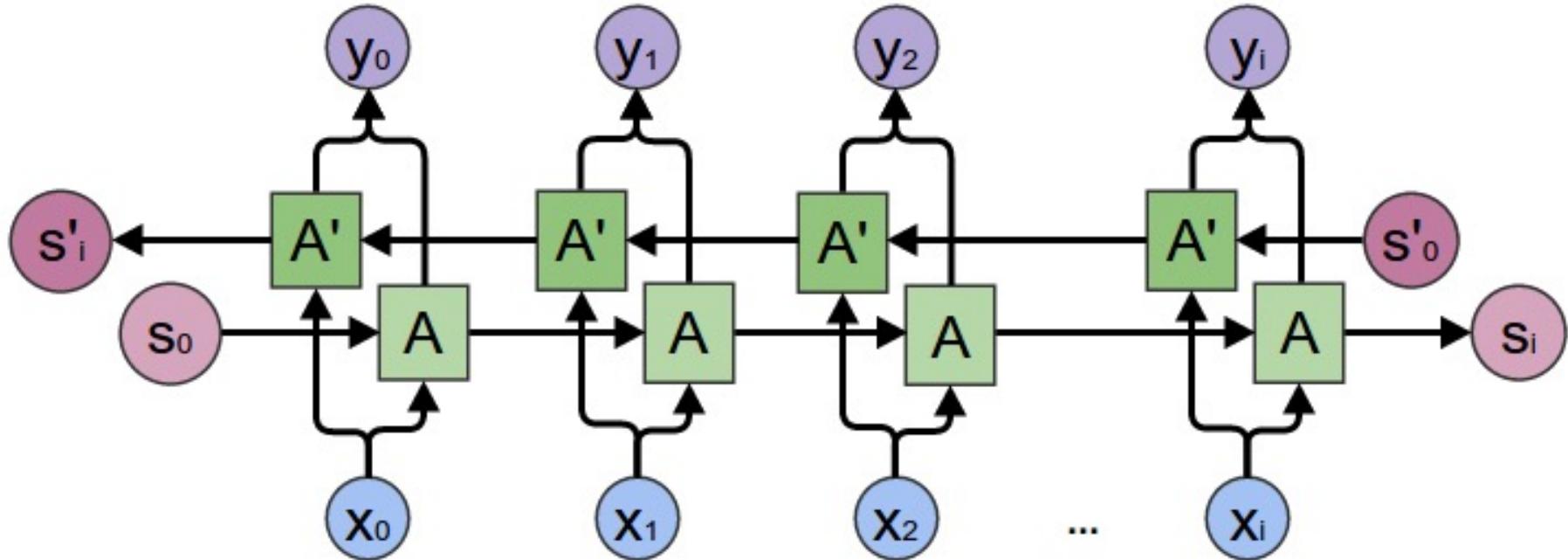
The **output gate** compute the output  $h_t$  to be passed to the next cell



$$o_t = \sigma(W_o[h_{t-1}, x_t]) \quad (20)$$

$$h_t = o_t \tanh(c_t) \quad (21)$$

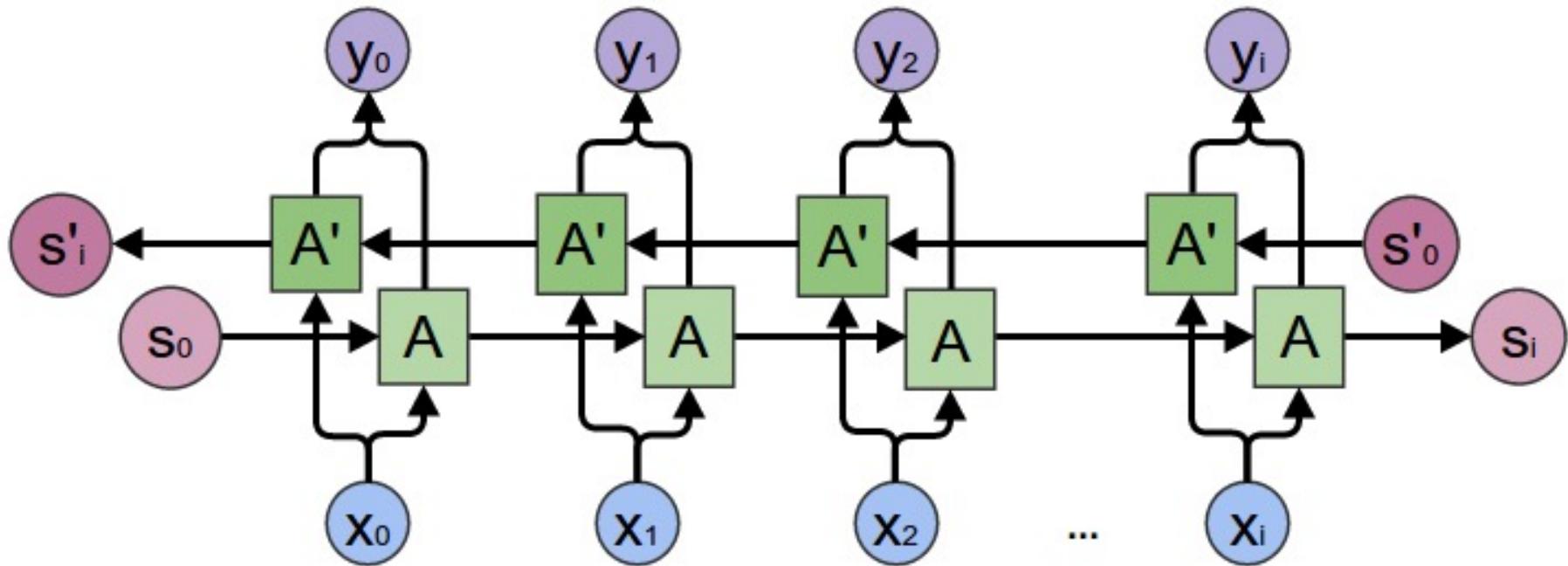
# RNN bi-directionnels



Un réseau dans chaque direction (passé->futur : A, et futur->passé : A')

- Permet de prendre en compte l'intégralité de l'information passée ET future
- MAIS n'est plus causal et temps-réel

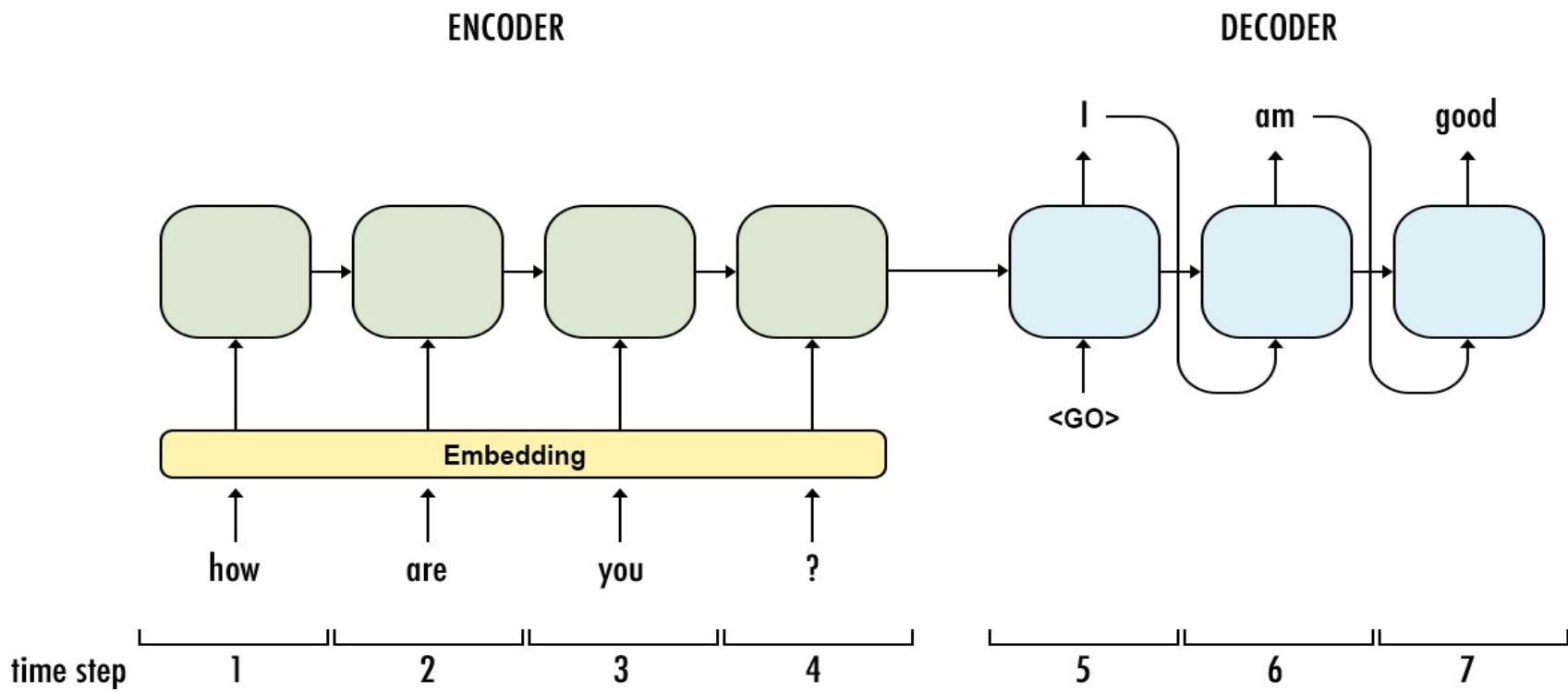
# RNN bi-directionnels



Opérateurs classiques pour combiner la mémoire du passé  $A$  et du futur  $A'$  :

- Concaténation
- Addition
- Multiplication point-à-point

# Modèle « séquence-à-séquence » (S2S)



- Apprendre une correspondance entre deux séquences de tailles variables ET différentes
- Applications pour le transcodage : traduction automatique, système de dialogue, reconnaissance/synthèse de la parole

# S2S à la vanille

[Sutskever, H. et al., 2014]

- On cherche à estimer une fonction  $f$  telle que,

$$y_1, \dots, y_{T_y} = f(x_1, \dots, x_{T_x})$$

- Pour se faire, on a besoin de calculer la probabilité conditionnelle,

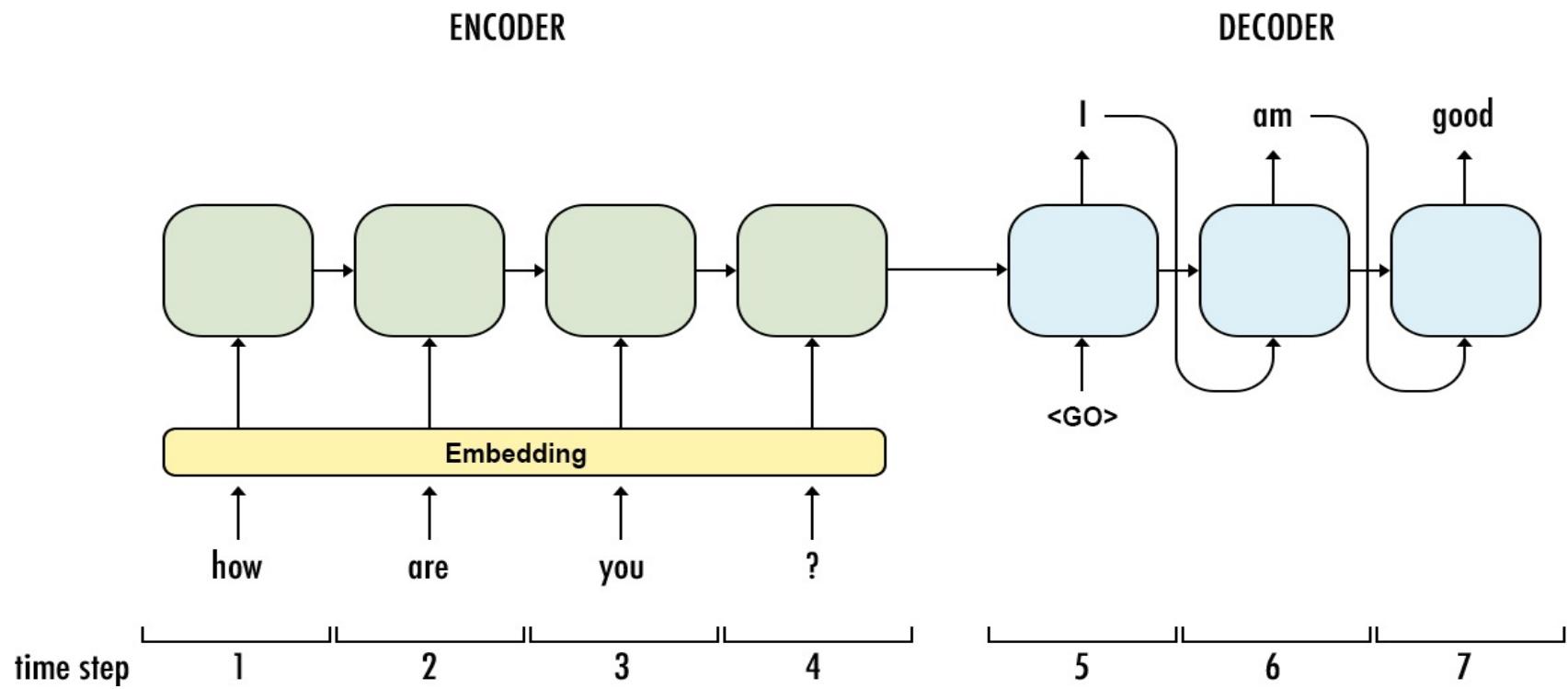
$$p(y_1, \dots, y_{T_y} | x_1, \dots, x_{T_x})$$

- Ce calcul n'est pas trivial !!!
- On divise ce problème en deux sous-parties résolvables,

$$x_1, \dots, x_{T_x} \rightarrow c \rightarrow p(y_1, \dots, y_{T_y} | c)$$

# S2S à la vanille

## [Sutskever, H. et al., 2014]



Une structure d' Encodeur- Décodeur RNN

$$\begin{aligned} & \frac{dy}{dx} = x^2 \\ & \frac{d^2y}{dx^2} = 2x \\ & \int_{t_0}^t dt = x \\ & f(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h} \end{aligned}$$

# S2S à la vanille

## [Sutskever, H. et al., 2014]

Encodeur RNN : encode une séquence de taille variable  $x$  par un code de contexte de taille fixe  $c$

- L'état  $h_j$  de la mémoire de l'encodeur RNN à l'instant  $j$ ,

$$h_j = f(x_j, h_{j-1})$$

- Le code  $c$  de taille fixe (c.a.d., indépendant du temps),

$$c = q(h_1, \dots, h_T)$$

- Typiquement, on ne conserve que le dernier état de la mémoire de l'encodeur,

$$c = h_T$$

# S2S à la vanille

## [Sutskever, H. et al., 2014]

Décodeur RNN : décode un code de taille fixe  $c$  par séquence de taille variable  $\mathbf{y}$

- On cherche à déterminer,

$$p(\mathbf{y}) = \prod_{i=1}^{T_y} p(y_i | y_{<i}, \boxed{c})$$

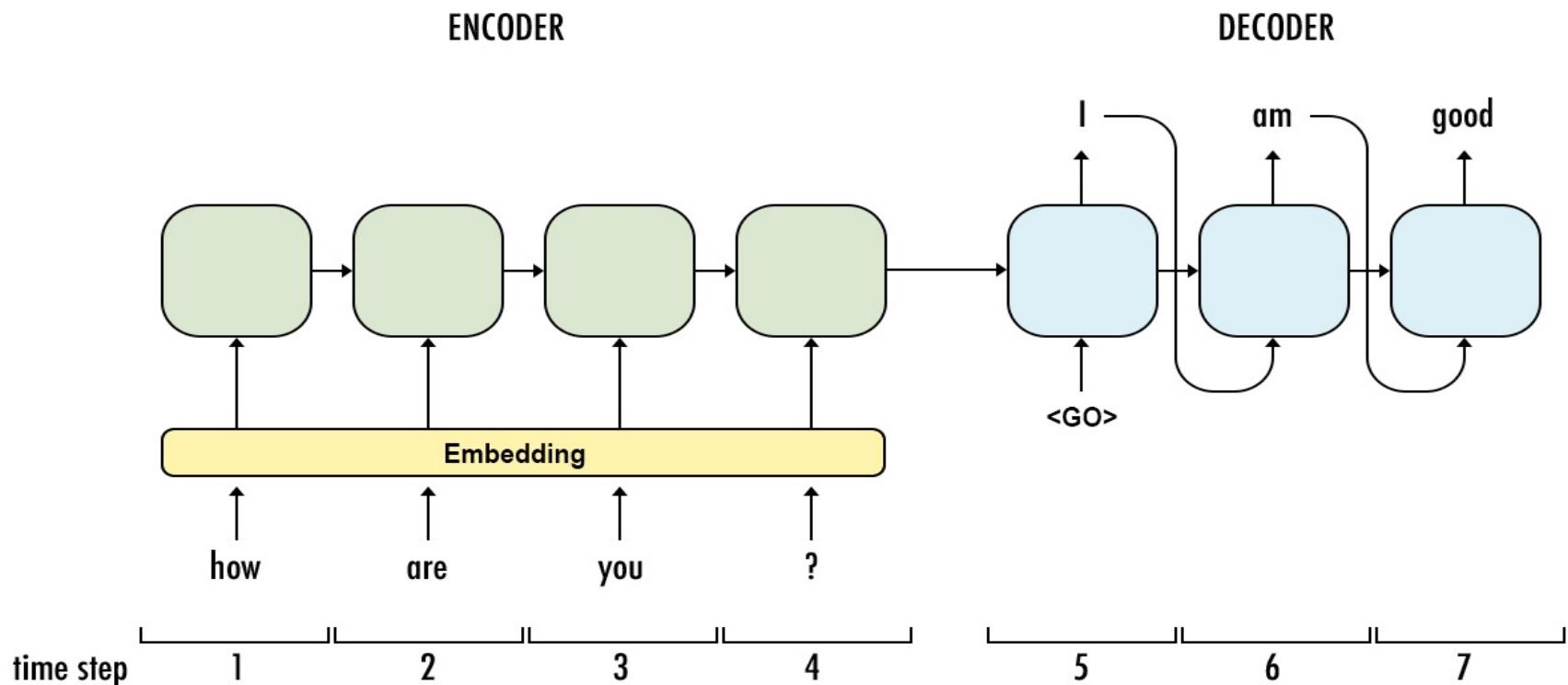
- On peut encore écrire cette probabilité sous forme d'un RNN,

$$p(y_i | y_{<i}) = f(y_{i-1}, s_i, c)$$

avec  $f$  la fonction d'activation,  $y_{i-1}$  la valeur de  $y$  à l'instant  $i - 1$  et  $s_i$  l'état de la mémoire RNN à l'instant  $j$

# S2S à la vanille

## [Sutskever, H. et al., 2014]



- Exemple d'inférence à partir d'un S2S

Handwritten mathematical notes and diagrams:

- A graph of a function  $y = f(x)$  with a vertical tangent line at a point on the curve.
- A derivative calculation:  $\frac{dy}{dx} = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h}$ .
- A geometric diagram showing a square divided into four triangles, with one triangle shaded.
- A formula:  $f(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h}$ .
- A fraction:  $AD/AB =$ .
- A complex fraction:  $\frac{(2x-5)^3}{x^2-3x-8}$ .

# S2S avec attention

## [Bahdanau, D. et al., 2015]

- Une autre manière permet d'écrire directement,

$$p(y_1, \dots, y_{T_y} | x_1, \dots, x_{T_x})$$

- Pour se faire, on introduit un mécanisme d'attention dans le décodeur RNN,

$$p(y_i | y_{<i}, \boxed{(x_1, \dots, x_{T_x})}) = f(y_{i-1}, s_i, \boxed{c_i})$$

- Le changement important est que maintenant le vecteur  $c_i$  dépend de la position  $i$  dans la séquence de sortie

# S2S avec attention

## [Bahdanau, D. et al., 2015]

- Le vecteur de contexte  $c_i$  est une simple combinaison de l'ensemble des états  $h_j$  de l'encodeur à tous les instants  $j$

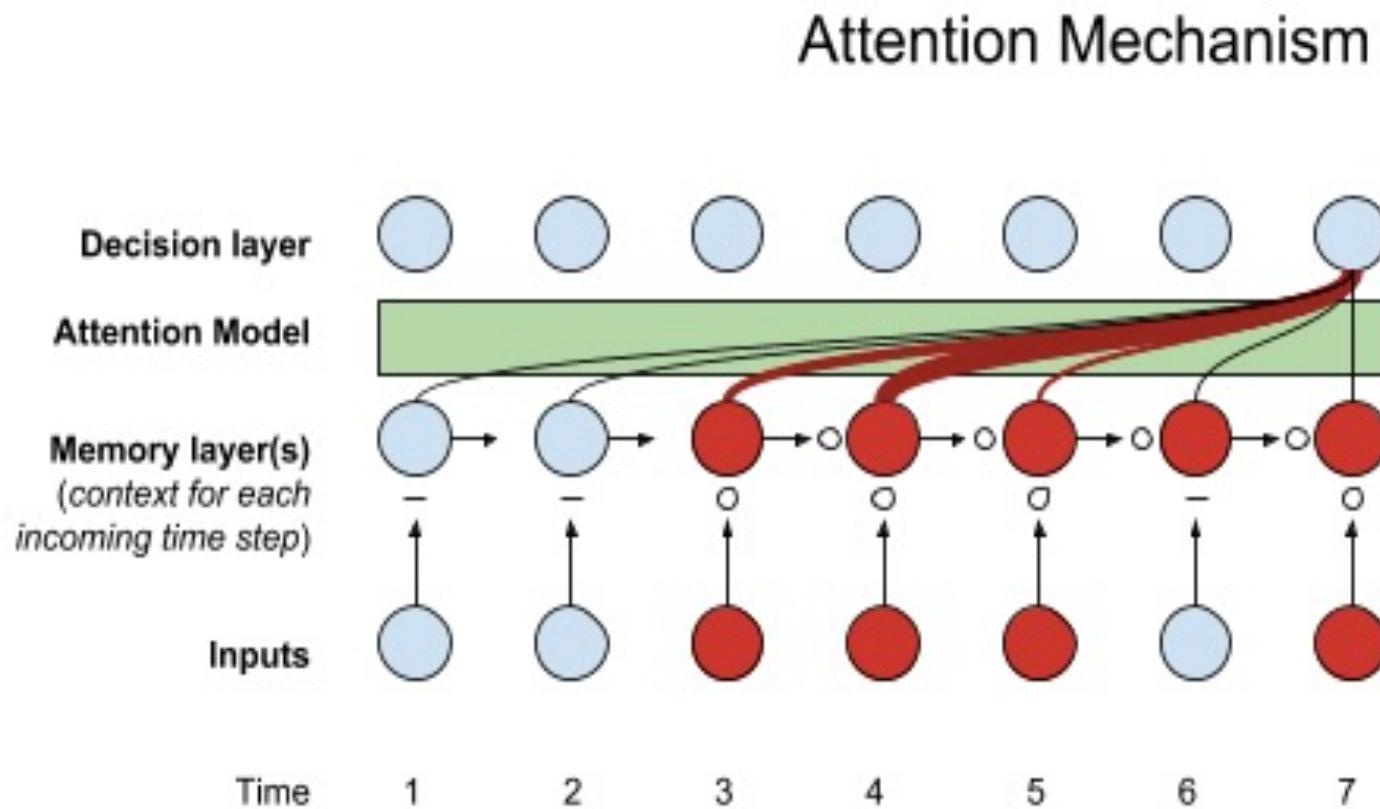
$$c_i = \sum_{j=1}^{T_x} \alpha_{i,j} h_j$$

- Le mécanisme d'attention de choisir l'information sur la séquence d'entrée  $j$  pertinente pour la prédiction à chaque temps  $i$  du décodeur
- De ce fait, l'attention réalise un alignement implicite entre la séquence d'entrée et la séquence de sortie

# S2S avec attention

## [Bahdanau, D. et al., 2015]

Le mécanisme d'attention de choisir l'information sur la séquence d'entrée  $j$  pertinente pour la prédiction à chaque temps  $i$  du décodeur



# S2S avec attention

## [Bahdanau, D. et al., 2015]

- Le **vecteur de contexte**  $c_i$  est une simple combinaison de l'ensemble des états  $h_j$  de l'encodeur à tous les instants  $j$

$$c_i = \sum_{j=1}^{T_x} \alpha_{i,j} h_j$$

- Les **poids d'attention**  $\alpha_{i,j}$  sont calculés à partir des scores d'attention,

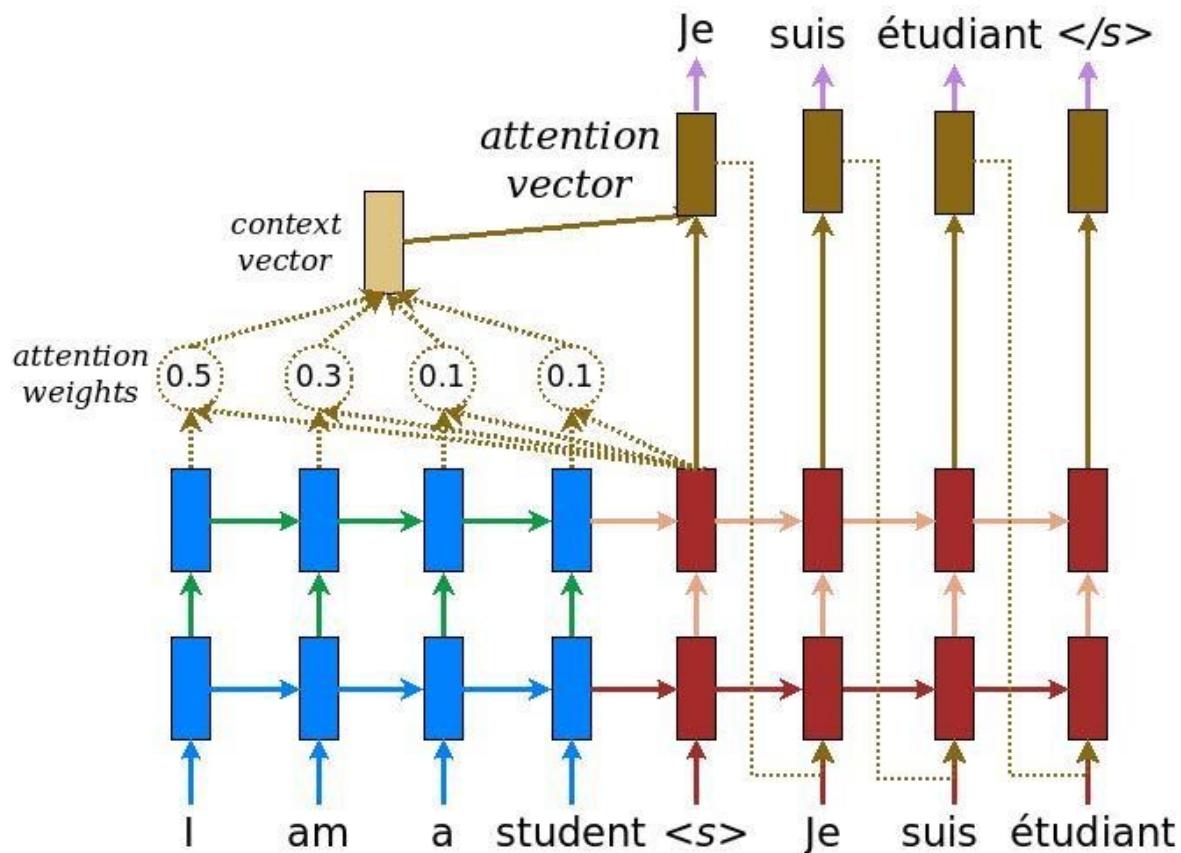
$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{j'=1}^{T_x} \exp(e_{i,j'})}$$

- Les **scores d'attention**  $e_{i,j}$  sont calculés par comparaison des états de la mémoire de l'encodeur et du décodeur (matrice de similarité)

$$e_{i,j} = a(s_{i-1}, h_j)$$

# S2S avec attention

## [Bahdanau, D. et al., 2015]



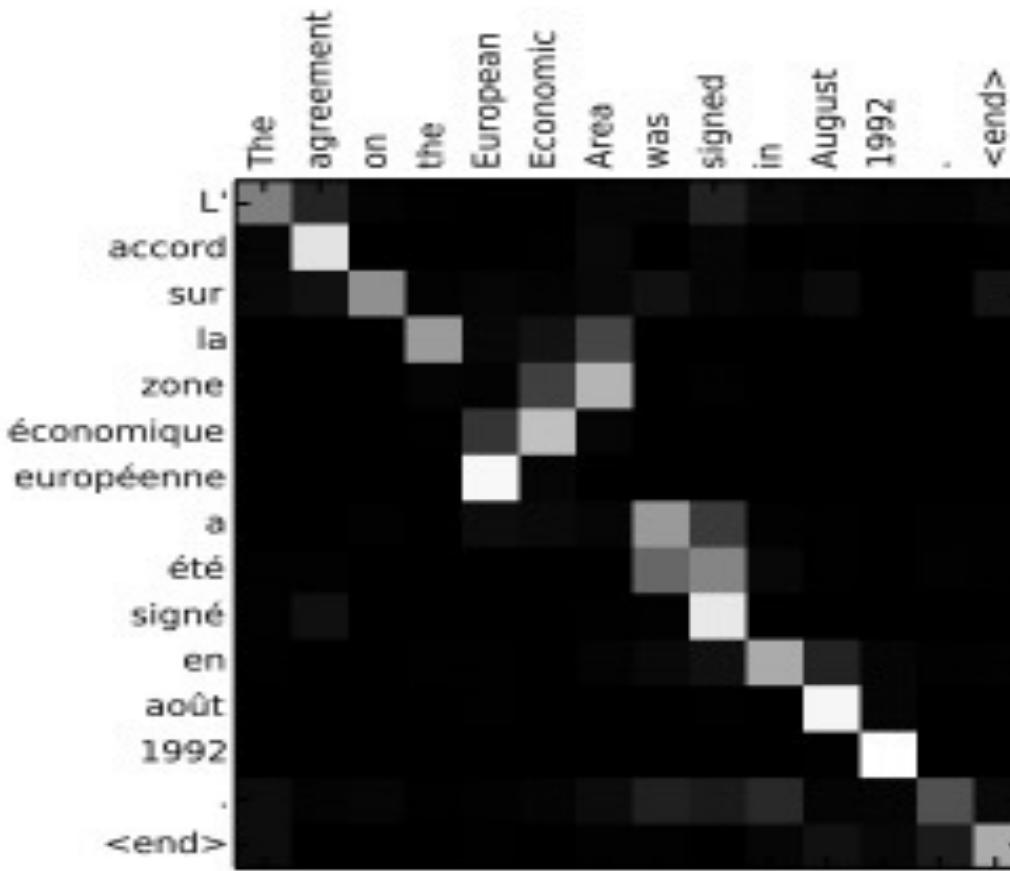
Poids d'attention  
score entre les encodages des mots target ht et des mots source hs

Vecteur de contexte  
somme pondérées des encodages hs avec leur poids d'attention alpha\_t,s

Vecteur d'attention  
Prédiction de la sortie à partir du vecteur de contexte ct et de l'encodage ht

# S2S avec attention [Bahdanau, D. et al., 2015]

Visualisation des poids d'attention pour un exemple de traduction français-anglais



# Attention!

- Le mécanisme d'attention est un concept général qui ne se limite pas nécessairement qu'aux réseaux récurrents!
- Exemple : On considère 3 maisons, chacune avec une **valeur** (un prix en €) et un ensemble **d'attributs** (par exemple, sa superficie, son nombre de pièces, et la présence d'une piscine)
- Pour inférer la valeur d'une nouvelle maison, on peut la déduire par **combinaison linéaire des valeurs** des maisons connues en fonction de leur **similarité** avec la nouvelle maison (en termes d'attributs)

Par exemple,

$$V = 0.6V_1 + 0.3V_2 + 0.1V_3$$

où:  $V_1$ ,  $V_2$ , et  $V_3$  sont les prix des 3 maisons et 0.6, 0.3, et 0.1 sont les poids correspondants à leur similarité avec la maison considérée.

# Attention!

De manière générale, on notera :

- (**k,v**) : les pairs de **clé** k et de **valeurs** v correspondant aux attributs de l'instance d'un objet et sa valeur
- **q** : la **requête** q correspondant aux attributs d'une nouvelle instance d'un objet, dont on cherche à connaître la valeur
- Nous pouvons alors écrire l'attention de manière générale comme,
$$\text{Attention}(q, k, v) = v \ S(q, k)$$
- Avec S est la fonction de score qui permet de mesurer la similarité entre les attributs de la clé et de la requête.

# Attention!

Fonctions de score usuelles :

- Le produit scalaire (pas de paramètres entraînables)

$$S(q, k) = q^T k$$

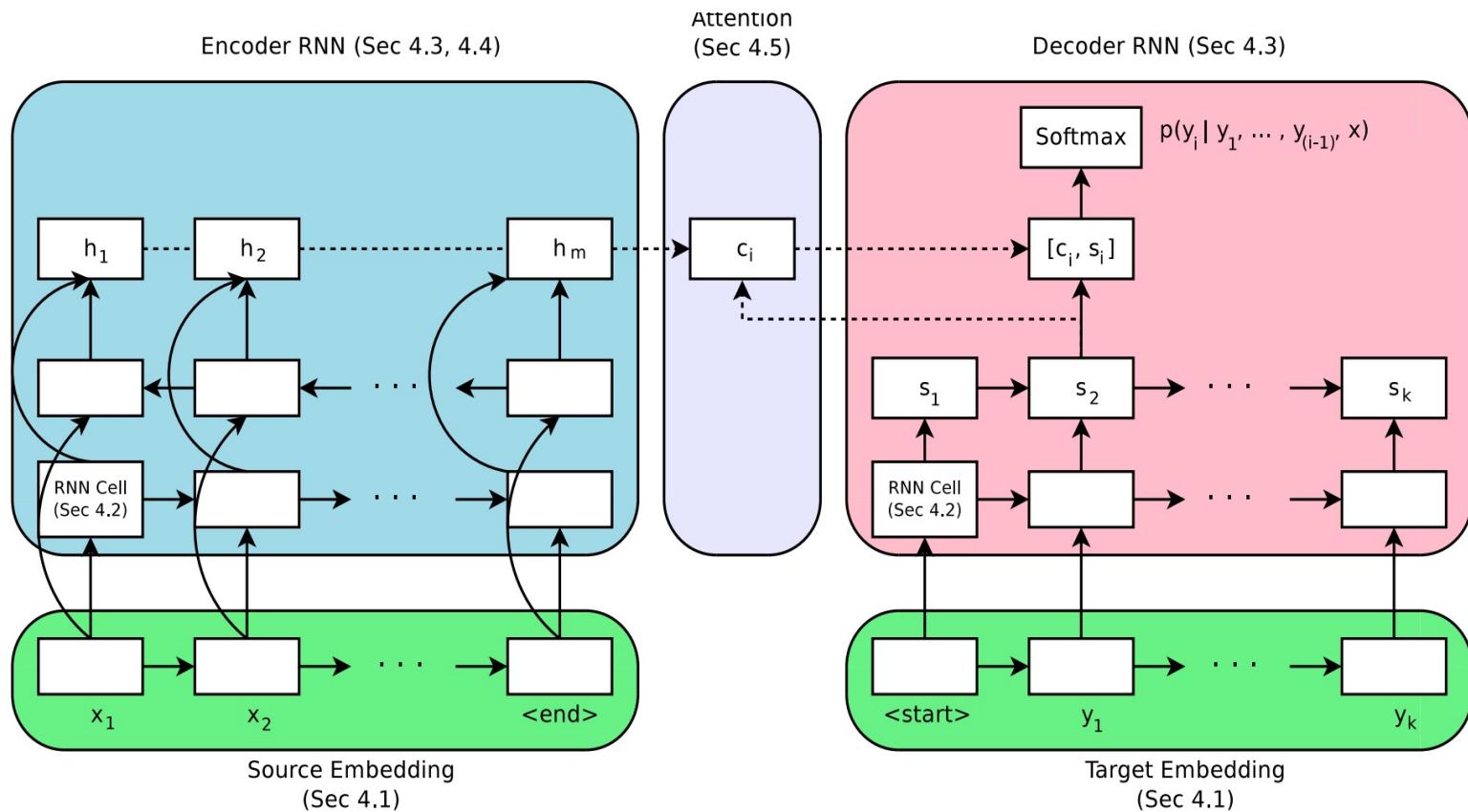
- Le produit scalaire normalisé (pas de paramètres entraînables)

$$S(q, k) = \frac{q^T k}{scaling\ factor}$$

- Une couche NN (de paramètres W)

# Attention!

Exercice : Identifier les clés, valeurs, et requête dans le modèle S2S



# Attention is all you need?

[Vaswani, A et al., 2017]

Si un mécanisme d'attention permet de traiter efficacement l'information temporelle

- À quoi servent encore les couches RNNs???
- Complexité algorithmique
- Non distribuables
- Mémoire temporelle limitée (inégale du plus proche au plus lointain)

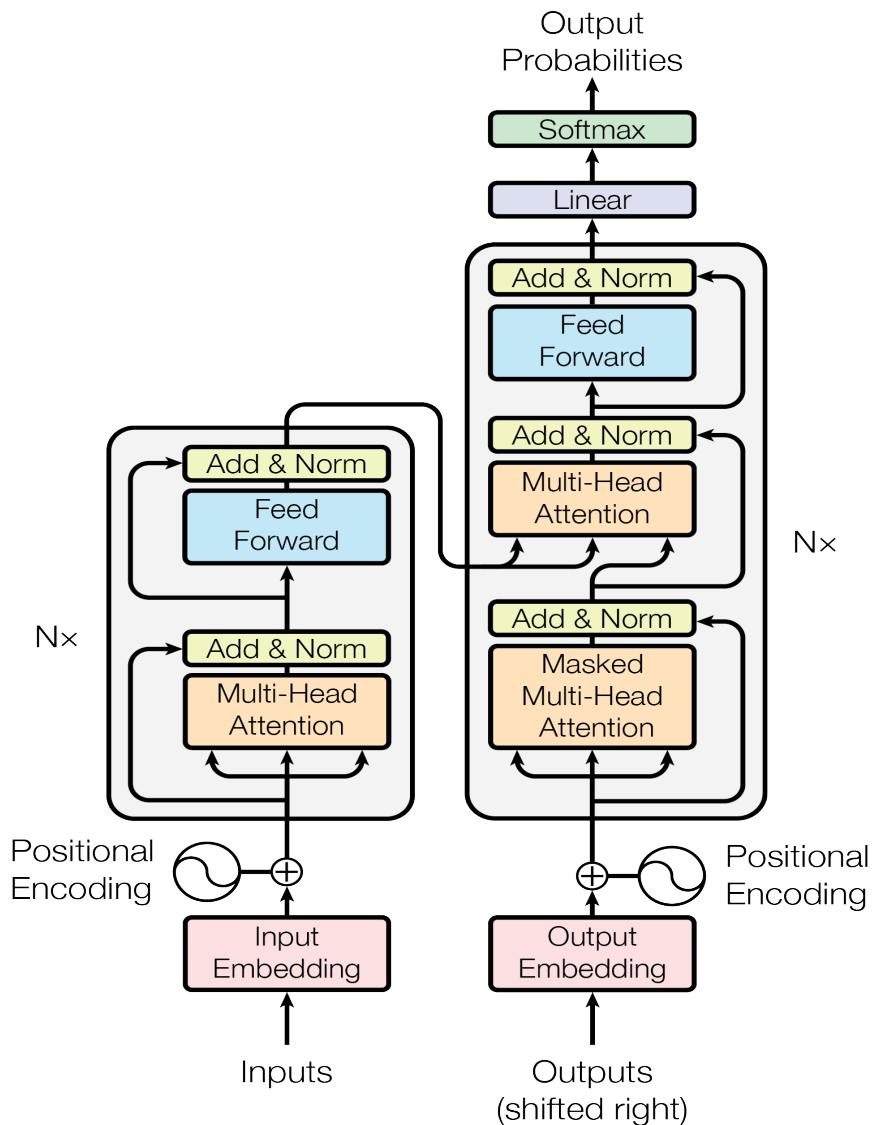


# Attention is all you need?

## [Vaswani, A et al., 2017]

Les réseaux « transformers » réalisent la modélisation de S2S à partir de :

- mécanisme d'auto-attention et multi-tête (« multi-head self-attention »)
- un encodage de la position dans la séquence
- Connexion résiduelles + batch norm.



# Attention is all you need? [Vaswani, A et al., 2017]

Dans un mécanisme d'**auto-attention** : les clés, valeurs, et requêtes sont identiques

- L'auto-attention apprend l'organisation interne de la séquence

## Encodeur

A chaque couche, les arguments de l'auto-attention sont les sorties de la couche précédente (incluant l'encodage de la position)

## Décodeur

A chaque couche, les arguments de l'auto-attention sont les sorties de la couche précédente (incluant l'encodage de la position)

## Interface Encodeur - Décodeur

Le mécanisme d'attention est classique : les clés et valeurs proviennent de la dernière couche de l'encodeur, la requête provient de la précédente couche du décodeur

# Attention is all you need?

## [Vaswani, A et al., 2017]

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types.  $n$  is the sequence length,  $d$  is the representation dimension,  $k$  is the kernel size of convolutions and  $r$  the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

### Avantages

- Calcul distribuable pour chaque couche
- Chaque élément de la série ne sont distants que d'un élément! Le traitement de l'information est indépendant de l'éloignement en temps

### Désavantages

- Complexité en  $n^2 \cdot d$  (ou  $r \cdot n \cdot d$  avec une attention réduite de longueur  $r$ )



# C7. Réseaux de neurones récurrents

Advanced Machine Learning (MLA)  
M2 Engineering of Intelligent Systems  
& Advanced Systems and Robotics

2020-2021